Technical Report

491

Sequential Decoding with a Small Digital Computer

I. Richer

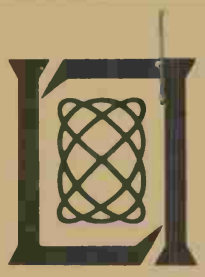
24 January 1972

Prepared for the Department of the Navy
under Electronic Systems Division Contract F19628-70-C-0230 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts



Approved for public release; distribution unlimited.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

SEQUENTIAL DECODING WITH A SMALL DIGITAL COMPUTER

I. RICHER

Group 66

TECHNICAL REPORT 491

24 JANUARY 1972

LEXINGTON MASSACHUSETTS

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Department of the Navy under Air Force Contract F19628-70-C-0230.

ABSTRACT

Extensive simulations of a sequential decoder using the Zigangirov-Jelinek algorithm have been conducted on a small, general-purpose digital computer. These simulations prove that this type of computer has sufficient memory, sufficient speed, and sufficient flexibility to perform sequential decoding at useful data rates.

In this report, the memory and computational requirements of the algorithm are presented, and efficient methods for ensuring a very low probability of error at any signal-to-noise ratio (at the expense of an increase in the failure-to-decode probability) are discussed. The equations necessary to set up a decoder are given, and a number of possible computer implementations are suggested.

Accepted for the Air Force
Joseph R. Waterman, Lt. Col., USAF
Chief, Lincoln Laboratory Project Office

CONTENTS

SEQUENTIAL DECODING WITH A SMALL DIGITAL COMPUTER

I. INTRODUCTION

Sequential decoding provides a highly reliable and efficient method for communicating at a low signal-to-noise ratio (SNR).[1] Generally, a sequential decoder is implemented either with special-purpose hardware or on a large digital computer. Earlier work with sequential decoders at Lincoln Laboratory in fact used special-purpose hardware in order to achieve data rates of several kilobits/second.[2,3] With the appearance over the past few years of a wide variety of small, low-cost, general-purpose computers, however, it was hoped that sequential decoding could be accomplished on such a machine at lower but still useful data rates. In this report we treat the problem of decoding efficiently and very reliably and show that, indeed, typical small (16-bit) computers possess sufficient memory, sufficient speed, and sufficient flexibility to perform the decoding task.

For definiteness, we stipulate that:

(a) The nominal SNR – i.e., the SNR corresponding to the nominal operating point – must be as low as possible, consistent with practical limitations on necessary computer speed and storage.

(b) The number of information bits N per block is small, say $N \sim 100$ bits.

(c) The quit probability P_Q – i.e., the probability that a block cannot be decoded – is low, say $P_Q \sim 10^{-3}$, at the nominal operating point.

(d) The probability P_E of decoding a block incorrectly is very low, say $P_E < \sim 10^{-7}$ at any SNR.

Note that because, in general, P_E increases as the SNR decreases, some margin must be incorporated in the nominal operating point to allow for possible changes in the channel. However, because of (a) this margin must be held to a minimum; the decoder itself should, to a large extent, adjust its performance to accommodate variations in the SNR, increasing P_Q in order to ensure that P_E does not exceed the specification.

We shall show that a data rate on the order of one kilobit/sec is possible with a decoder that meets the above requirements and that operates in real time on a present-day small computer; for a binary antipodal additive Gaussian noise channel with continuous output, this data rate can be realized with E_b/N_o – that is, the ratio of the received energy per information bit to the single-sided noise power density – less than 2.8 dB. Furthermore, even higher data rates are possible if the requirements are relaxed slightly.

Currently, there are two algorithms for sequential decoding in general use: the Fano algorithm,[1] in which both forward and backward progress in the decoding tree is made one node at a time, and the Zigangirov-Jelinek (henceforward referred to as ZJ) algorithm,[4,5] which saves information on paths that have been examined for possible use at a later time. In contrast with the Fano algorithm, the ZJ algorithm never repeats a computation, so there is a trade-off between the smaller number of computations of the ZJ and the smaller memory requirements of the Fano. In light of the potential speed advantage of the ZJ algorithm and the relatively low (and continually decreasing) cost of computer memory, the ZJ algorithm is of primary interest here. However,

when this study was initiated, no quantitative information was available on the performance of the ZJ algorithm. Since quantitative results are difficult to obtain analytically, simulations were performed. The results presented in this report are based entirely upon these simulations. Although a binary, additive white Gaussian noise channel was used in the simulations, most of the results are directly applicable to other channels. The following section gives some background material and the equations necessary for setting up a decoder. Section III describes the ZJ algorithm and presents the computational and memory requirements. The error performance of the decoder, and methods for limiting P_E regardless of the SNR, are discussed in Section IV. Section V describes in detail the implementation of the ZJ algorithm, and the final section presents conclusions and suggestions for possible implementations on a small computer. Because the characteristics of the simulated noise source can have a profound effect upon the results, considerable effort was devoted to developing and testing programs for producing computer-generated random numbers. The algorithm employed in the simulations is detailed in Appendix A.

II. BACKGROUND

Detailed derivations of the theory and operations of convolutional encoders and sequential decoders are available in the literature.[1,6] In this section, therefore, we shall only summarize the salient characteristics of the encoder/decoder, with emphasis on the equations pertinent to implementing a decoder on a digital computer.

A. Qualitative Discussion

In general, a binary convolutional encoder is made up of a K-stage shift register and V parity networks. Each parity network generates the modulo-2 sum of its binary inputs. An example of an encoder with K = 4, V = 3 is shown in Fig. 1. Initially the shift register is cleared to zero (or set to any known state) and the first information bit is fed in. The parity networks are then sampled in succession, producing V output bits or channel symbols. The rate R of a binary encoder is defined as the ratio of the number of input and output bits and is therefore given by R = 1/V. The next information bit is then fed into the shift register, the previous contents being shifted one place, and the next set of V channel symbols are produced. This procedure continues for the duration of the information sequence. Each information bit remains in the shift register for K shifts, and hence K is referred to as the constraint length of the code. The reliability of convolutionally encoded messages arises because KV channel symbols are affected by each information bit. In order to provide the necessary redundancy on the final information bits, a tail of K − 1 zeros (or any other known sequence) is appended to the information sequence and passed into the shift register. Thus, for N information bits a total of (N + K − 1) V channel symbols is transmitted. The penalty for providing this tail is an increase in the transmitter power (or in the message transmission time) by the factor (N + K − 1)/N. For N = 100 and K = 30, the penalty is 1.1 dB.

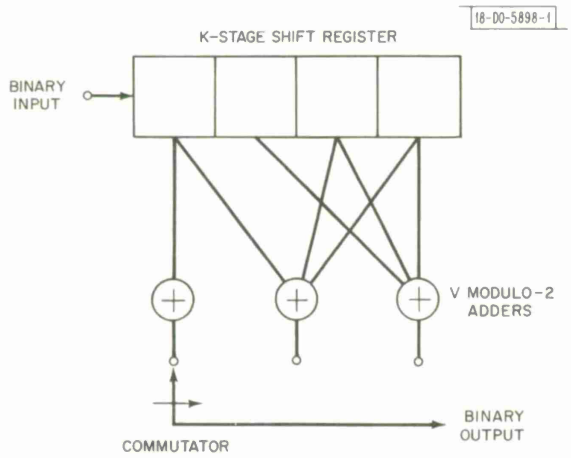


Fig. 1. Binary convolutional encoder with K = 4, V = 3.

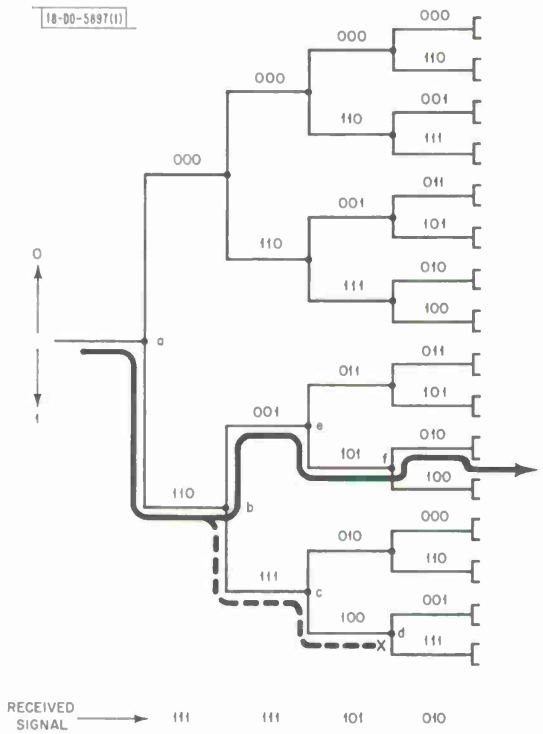


Fig. 2(a). Code tree for encoder of Fig. 1.

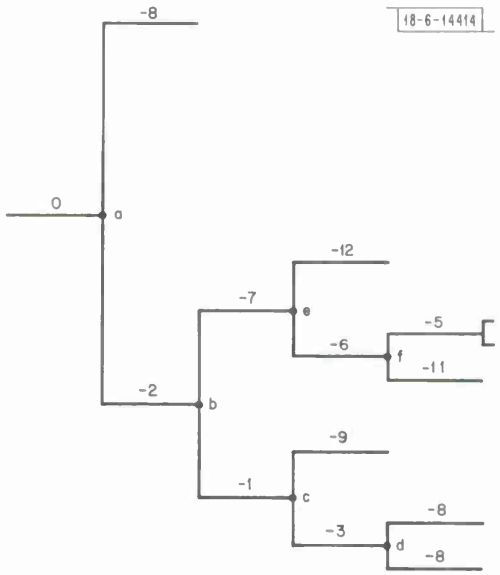


Fig. 2(b). Metrics for the nodes along the paths marked in (a). The metrics were computed from the expression [1 – 3 × (no. disagreements)] for the received signals shown in (a).

The encoder output is conveniently represented by a binary code tree, as shown in Fig. 2(a). At each node in this tree, the upper branch is taken if the next information bit is a zero, and the lower branch if it is a one. The input sequence 1010... thus produces the output sequence 110001101010... .

The decoder contains a replica of the convolutional encoder and using the received channel symbols attempts to recover the transmitted message by finding the most likely path through the code tree. But there are 2^N paths, so for $N \gtrsim 30$ it is impossible for a decoder to examine them all within a reasonable time. A sequential decoder avoids this problem by restricting its examination to those paths which appear most promising. The decoder starts at the beginning of the tree and computes the expected received signals if the first information bit were a zero and if it were a one. Then, based upon the actual received signals, the decoder makes a tentative decision on the first bit, and advances along the appropriate branch to the next node in the tree. The tentative decision made at this and all subsequent nodes is determined from the a priori probability that the sequence of received signals corresponds to the coded bits associated with the branches stemming from the node; the numerical quantity based upon this probability is called the metric. The metric associated with a given node in the tree is the sum of the branch metrics along the path from the start of the tree to that node. On the average, the metric increases along the correct path and decreases along incorrect paths. During periods of high noise, however, the metric along the correct path may decrease. If the metric for the path under investigation does decrease sufficiently – either because the path is incorrect or because of noise on the correct path – the

decoder will reverse an earlier branch decision and attempt to find a better path through the tree.

As an illustration, assume that the received signals are quantized as zeros or ones (i.e., hard decisions) as shown in Fig. 2(a), and assume that the decoder parameters are such that the metric increment for a branch be 1 − 3D, where D is the number of differences between the expected bits and the received bits. Initially, the dashed path indicated in Fig. 2(a) appears attractive because its metric takes on the successive values −2, −1, −3, as shown in Fig. 2(b). At node d, however, the metrics for both branches decrease to −8. Because of the decreasing metric along segment bcd..., the decoder will backtrack, eventually returning to node b and progressing forward with an increasing metric along segment bef. Some more realistic data are illustrated by the curves in Fig. 3. These computer-generated curves show the metric as a function of depth (level) in the decoding tree for two simulated 100-bit messages.* The decoder had

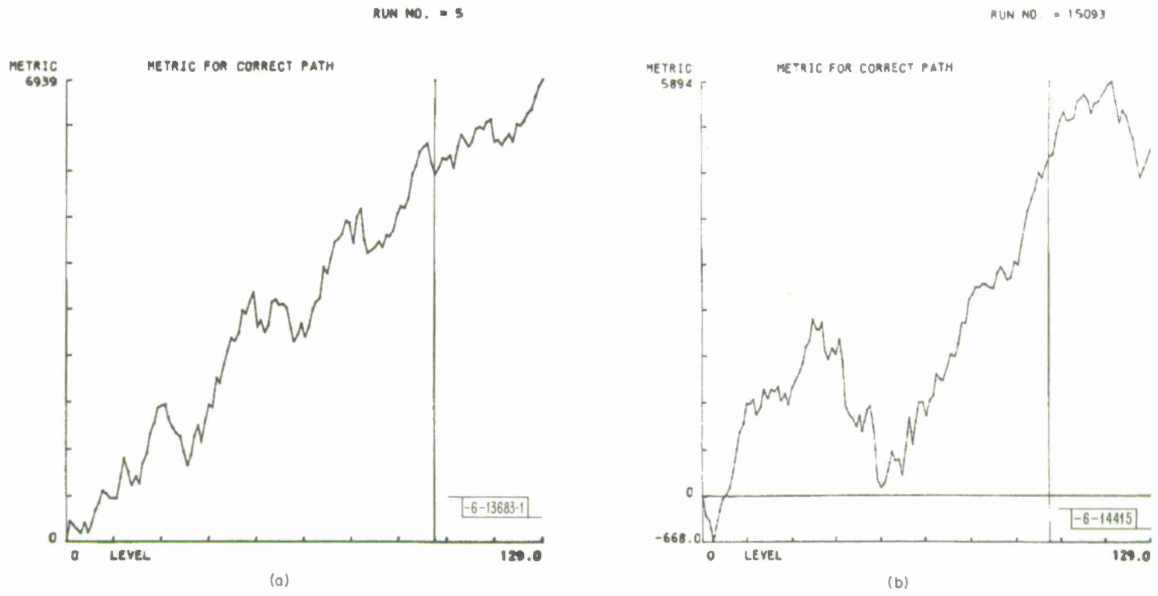


Fig. 3. Computer-generated curves of the metric (along the correct path) for two simulated messages using Gaussian noise, with N = 100, K = 30 (and R = R_{comp}). The vertical line in each figure indicates the start of the tail. Part (a) shows the metric for a message that was easily decoded, whereas (b) is for a message that was relatively noisy and hence difficult to decode.

no difficulty decoding the message of Fig. 3(a) because the relatively noisy periods, manifested by the shallow troughs in the metric curve, were spaced well apart and were of short duration. The depth and width of the trough in Fig. 3(b), however, forced the decoder to search many more incorrect paths before singling out the correct one.

B. Quantitative Discussion

The preceding qualitative description of the operation of a sequential decoder will now be quantified by the presentation of some pertinent analytical results. A decoder computation is defined as the process of moving from the node last examined to the node that will be next examined. This process includes the operations required to calculate the two branch metrics

* The channel was represented by white Gaussian noise, and the metrics were based upon Eq. (8).

from a node and to execute the decisions based upon those metric values. If C_{tot} is the total number of computations needed to decode a message of N bits, then the average number of computations per information bit for that block is $C \equiv C_{tot}/N$, and the distribution of C is approximately of the Pareto form

$$P[C \geq X] \approx AX^{-\alpha} \tag{1}$$

for large X, the quantities A and α depending primarily upon the channel.[1,6] For $\alpha < 1$, the expected value of the number of computations per bit E(C) exists, but for $\alpha > 1$, E(C) diverges. The value of α is thus of importance in determining decoder operation, and the value $\alpha = 1$ is of critical importance.

In practice, of course, decoding is terminated (with the block undecoded) after some specified time, so E(C) always remains finite; the practical implication of Eq. (1) is that the quit probability increases sharply for $\alpha > 1$. The decoder rate for which $\alpha = 1$ is called the computation cutoff rate and for equally likely binary inputs is given by

$$R_{comp} = 1 - \log_2 \left[1 + \int_{-\infty}^{\infty} \sqrt{w(y|0)\, w(y|1)}\, dy \right] , \tag{2}$$

where $w(y|i)$ is the probability density of the received signal y, given that the symbol i is transmitted (i = 0, 1). The channel is represented by additive white Gaussian noise having single-sided noise power density N_o. Then, for binary antipodal modulation* the expected value of the received signal at the output of the matched filter (i.e., at the input to the decoder) is $+\sqrt{2E_s}$ for a transmitted 1-bit and $-\sqrt{2E_s}$ for a 0-bit, and

$$w(y|i) = \frac{1}{\sqrt{2\pi N_o}} \exp[-(y \pm \sqrt{2E_s})^2/2N_o] .$$

The integral in Eq. (2) now becomes

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp[-(t^2 + a^2)/2]dt = e^{-a^2/2} ,$$

yielding

$$R_{comp} = 1 - \log_2[1 + e^{-a^2/2}] , \tag{3}$$

in which

$$a = \sqrt{2E_s/N_o} . \tag{4}$$

Note that E_b, the energy per information bit, is $E_b = VE_s$ (neglecting the tail), so that

$$\frac{E_b}{N_o} = V\frac{a^2}{2} . \tag{5}$$

* Binary antipodal signaling, together with coherent detection, yields a communications structure that is highly efficient in terms of required SNR.[7]

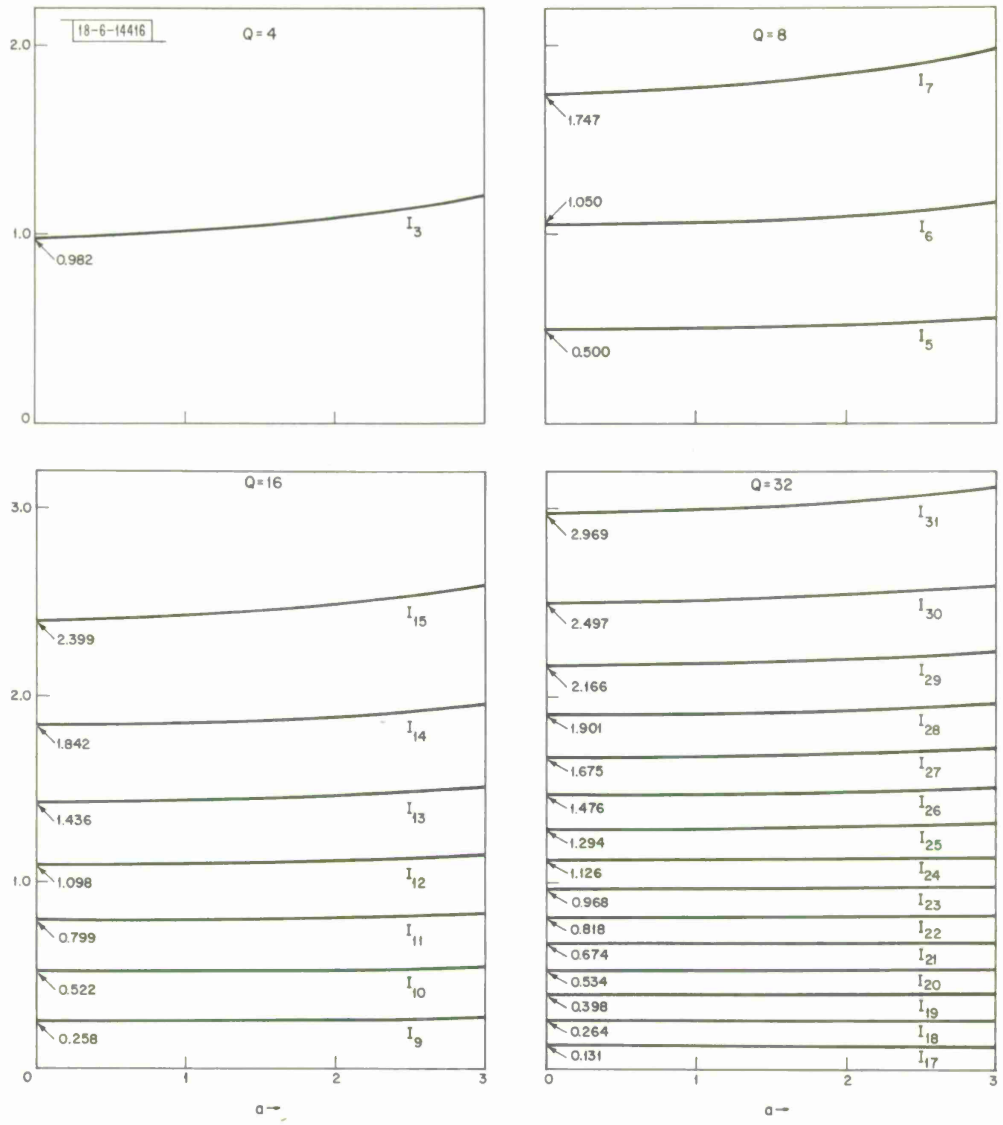


Fig. 4. Boundaries of the optimum quantization intervals as a function of $a = \sqrt{2E_s/N_o}$ for several values of Q. The intervals are chosen symmetrically about 0.

The additional signal power needed to transmit L tail bits is given by the factor (1 + L/N). At $R = R_{comp}$, Eqs. (3), (4), and (5) may be combined to yield the expression

$$\frac{E_b}{N_o} = -V \ln(2^{1-(1/V)} - 1) \quad , \quad R = 1/V = R_{comp} \quad , \tag{5a}$$

which relates the required input signal strength to the number of parity networks.

Equations (2) and (3) assume that the decoder has available the received signals to full precision, whereas on a digital computer these signals are quantized. Since the decoder cannot distinguish between signals that fall in the same quantization interval, performance will be degraded, the degradation evinced by a decrease in R_{comp}. If q_{ij} is the probability that the received signal falls in the j^{th} quantization interval ($j = 1, 2, \ldots, Q$), given that the symbol i is transmitted ($i = 0, 1$), then the integral in Eq. (2) is replaced by a sum, and R_{comp} becomes

$$R_{comp} = 1 - \log_2 \left[1 + \sum_{j=1}^{Q} \sqrt{q_{0j} q_{1j}} \right] \quad . \tag{6}$$

For a particular value of a, R_{comp} can be maximized by choosing the quantization intervals optimally. Because of the symmetry (about 0) of the received signals, symmetric quantization intervals are used. If we denote the upper endpoints of the j^{th} quantization interval by I_j, then the symmetry implies that

$$I_{Q-j} = -I_j \quad , \quad I_o \equiv -\infty \quad , \quad I_Q = +\infty \quad , \quad I_{Q/2} = 0$$

and the transition probabilities are

$$q_{ij} = \frac{1}{\sqrt{2\pi}} \int_{I_{j-1}}^{I_j} e^{-(t \pm a)^2/2} \, dt \quad , \quad \begin{cases} i = 0, 1 \\ j = 1, 2, \ldots, Q \end{cases} \quad . \tag{7}$$

In this equation the plus sign is used for i = 0 and the minus sign for i = 1. Figure 4 plots for Q = 4, 8, 16, and 32 the $\{I_j\}$, ($j = Q/2 + 1, \ldots, Q - 1$), that maximize R_{comp}. It is evident that the optimum $\{I_j\}$ are only weakly dependent upon a and hence upon the SNR. Further, the optimum intervals provide only a slight improvement in R_{comp} over uniformly spaced intervals. The latter data are given in Fig. 5, which also gives R_{comp} for no quantization ($Q = \infty$). (However, since total decoding time differs only negligibly if the optimum rather than the uniform intervals are employed, and since a small fraction of a dB can have a significant effect on decoder performance when operation is at a rate near R_{comp}, the optimum $\{I_j\}$ were used in the simulations.) The practical limitation on the size of Q is the amount of memory that can be devoted to storing incoming quantized signals.

The decoder metrics are computed from the likelihood ratio according to the formula

$$m'_{ij} = \log_2 \left[\frac{2q_{ij}}{q_{0j} + q_{1j}} \right] - U \quad . \tag{8}$$

Thus, m'_{ij} is the metric contribution if channel symbol j is received, given that i is transmitted. In this equation the term U, called the bias, permits a meaningful comparison between metrics at different depths in the decoding tree. The choice of U is important for efficient decoder operation: if it is too small, then metrics for some incorrect paths will increase, causing a

18-6-14417

$a = \sqrt{2E_s/N_o}$	NUMBER OF QUANTIZATION INTERVALS (Q)	R_{comp}	
		OPTIMUM INTERVALS	UNIFORM INTERVALS*
0.2	4	0.0127	–
	8	0.0139	0.0138
	16	0.0142	0.0141
	32	0.0143	0.0143
	∞	0.0144	–
0.5	4	0.0773	–
	8	0.0844	0.0841
	16	0.0865	0.0861
	32	0.0871	0.0870
	∞	0.0874	–
1.0	4	0.283	–
	8	0.306	0.305
	16	0.313	0.312
	32	0.315	0.315
	∞	0.316	–

*For uniform intervals the spacing used was

$$I_j - I_{j-1} = \begin{cases} 0.5_j & Q = 8 \\ 0.25_j & Q = 16 \\ 0.15_j & Q = 32 \end{cases}, \quad j = 2, \ldots, Q-1$$

Fig. 5. Values of R_{comp} for optimally spaced and for uniformly spaced intervals for several values of Q and a. $Q = \infty$ corresponds to no quantization.

decoder error; if it is too large, all metrics, including the correct one, will decrease, and too many computations will be performed. The optimum bias is close to the decoder rate R (Ref. 6).

As with the received signals, the metric values must also be quantized. For convenience, we allow only integral values, and for practical reasons we must restrict their range. Thus, the metric values are scaled to

$$m_{ij} = \mathrm{INT}(\lambda m'_{ij} + 1/2) \quad , \tag{9}$$

in which INT(x) is the integer part of x, and the (1/2) is included so that the values are properly rounded. The multiplicative factor λ is chosen according to the formula

$$\lambda = \frac{\text{(desired range of metrics)}}{|\text{largest } m'_{ij} - \text{smallest } m'_{ij}|} \tag{9a}$$

More specific details on the choice of λ (and of U) are given in the next section, where simulation results are presented.

Equations (4) through (9) provide the information necessary to calculate the parameters for a sequential decoder. A typical design might begin with the selection of a value for V. Since the required SNR decreases as V increases, a large V is desirable [cf. Eq. (5a)], but the actual choice of V is often limited by outside constraints such as the available bandwidth or the performance of a phase tracker. With regard to implementation on a 16-bit computer, we note that the decoder complexity increases somewhat if $V > 16$. However, Eq. (5a) reveals, for example, that $V = 12$ and $V = 6$ degrade E_b/N_o by only 0.13 dB and 0.27 dB, respectively, from the limiting value of 1.42 dB as $V \to \infty$ for unquantized signals. (Quantized signals yield similar degradation.[8]) For operation at the computation cutoff rate, Eqs. (6) and (7) are solved for a, in order that $R_{comp} = R = 1/V$. To minimize the degradation caused by quantization we choose $Q \geq 8$ and use the optimum $\{I_j\}$. After a (and hence the SNR) is determined, the metric increments can be computed from Eqs. (8) and (9). With $V = 12$ and $Q = 8$, for example, the required E_b/N_o is 1.7 dB plus an additional 1.1 dB for the tail if $N = 100$, $L = K - 1 = 29$.

III. PERFORMANCE OF THE ZIGANGIROV-JELINEK ALGORITHM

The ZJ algorithm for sequential decoding, proposed independently by Zigangirov[4] and by Jelinek,[5] utilizes the memory available on a digital computer to save results that may be required at a later time by the decoder. The algorithm thereby avoids repeating earlier computations but, as with many digital computer programs, the time saved is at the expense of increased memory requirements. Because of the steadily decreasing cost and increasing capacity of digital computer memories, this trade-off appears worthwhile. In this section we present simulation results showing that the memory requirements of the ZJ algorithm are in fact quite reasonable. We also give the computational requirements of the algorithm and the effect of various parameters upon decoder performance.

A. Parameter Values

In all the simulations discussed in this report, the following parameter values were used, unless there is a specific statement to the contrary:

N = number of information bits per block = 100 bits,

K = constraint length = 30 bits,

V = number of parity networks = 12,

Q = number of quantization intervals = 8 (chosen optimally),

U = metric bias = R = 1/V.

The experimental results given in this report are valid for any implementation of the ZJ algorithm that uses the same parameter values. Moreover, many of the results can be applied directly or with minor modifications to decoders with different parameter values (cf. Section III-E). In order to obviate the need for a convolution encoder, the all-zero message was assumed. Because of the group properties of the code, no loss of generality results from this assumption. However, in order to avoid a possible favorable bias in the decoder, the 1-branch is chosen if the two branches from a node have equal metrics. The channel was simulated by generating independent Gaussian noise samples to represent the output of the matched filter. The algorithm that was used to generate the samples is described in Appendix A.

B. The Basic ZJ Algorithm with Finite Memory

In contrast with the Fano algorithm, the ZJ algorithm is conceptually quite straightforward. A table is maintained in memory, with each entry on the table containing information on nodes in the decoding tree that have been examined. As outlined in the previous section, the metric associated with each node is a measure of the likelihood that the node is on the correct path; therefore, of all the nodes in the table, the decoder extends the path from the one having the largest metric, until the end of the tree is reached. More formally, the steps that comprise the ZJ algorithm are:

(1) Initialize by clearing the memory table and creating one entry corresponding to the start of the decoding tree.

(2) Retrieve the entry with the largest metric. If this entry is at the end of the tree, decoding is finished.

(3) Compute the two branch metrics stemming from the node found in Step (2) and create two entries in place of the original. Store these two entries on the table.

(4) If the table is full, discard the entry with the worst metric.

(5) Go to Step (2).

In principle, Step (4) is not part of the ZJ algorithm, but in practice it is required because only a limited amount of memory can be devoted to the memory table. If the table can accommodate T entries, then when the table first becomes filled (assuming that decoding has not been completed), the entries correspond to the first T nodes that were examined. Once the table is full, however, one entry must be eliminated for each node examined (unless the node extended in Step (3) is in the tail, in which case no new entry is generated). Since there is a finite number of nodes in the decoding tree, the process described by the above steps will eventually terminate.

C. Computation Cutoff

When a path is extended from depth d in the decoding tree to depth d + 1, the table entry corresponding to the node at depth d is eliminated [Step (3) above]. Thus, each node that is represented on the table is the one that is at the greatest depth on a path into the tree. In particular, there can be only one entry corresponding to the correct path. With a finite table size

there exists the possibility that during a period of high noise the correct-path entry will have the worst metric and will therefore be discarded. If this occurs, the decoder will experience difficulty in decoding, but if allowed sufficient time, it will eventually reach the end of the tree, and the decoded block will necessarily contain an error. In order to preclude this possibility, a cutoff must be imposed upon the amount of time spent decoding each block. Actually, such a cutoff is also necessary because incoming signals are continually being received, and (1) there is a limit on the amount of buffer storage that can be allocated for these signals while they wait to be processed, and (2) there may be a limit on the time delay that can be tolerated between the receipt and the decoding of a block. The cutoff that is used will be the minimum dictated by all these constraints.

The time consumed by the decoder is approximately proportional to the number of decoder computations that are performed, where a single decoder computation comprises all the operations required by Steps (2) to (5) of the algorithm. A cutoff expressed in terms of the number of computations is therefore equivalent to one expressed in terms of the real decoding time. The data presented in Section V on the time per computation can be used to relate the two cutoffs. Quantitative information on the computation cutoff is given in Section IV-A in connection with the error probability associated with a finite-memory ZJ decoder.

D. Computation Distribution and Quit Probability

The distribution of the average number of computations C per information bit is shown in Fig. 6(a). This curve is based upon 2×10^5 simulated messages,* with the decoder operating at rate $R = 0.99\, R_{comp}$. The distribution is Pareto, with the data fitting the equation

$$P(C \geq X) \approx (0.135)\, X^{-1.05} \quad .$$

The exponent is close to -1, consistent with theoretical predictions for operation at $R = R_{comp}$. The distribution as drawn applies to a ZJ decoder with unlimited memory. This was accomplished by using a table that was large enough so that, for the range of computations shown, the computation cutoff did not come into play. As the table size decreases, the distribution shifts upward because there is an increase in the probability of discarding the correct path, and hence an increase in the probability of reaching computation cutoff.

Figure 6(b) shows the quit probability P_Q as a function of table size T for the same simulated messages used in Fig. 6(a). This curve also exhibits Pareto behavior, obeying the empirical expression

$$P_Q(T) \approx (3.10)\, T^{-1.03} \quad .$$

It seems remarkable that a decoder with a table having only 200 entries is able to decode 99 percent of the blocks. On the other hand, the -1 exponent in the expression for $P_Q(T)$ implies that extremely low P_Q cannot practicably be realized at $R = R_{comp}$. Several methods for reducing $P_Q(T)$ were investigated in order to permit a reduction in the required memory at $R \sim R_{comp}$. Some of the methods proved quite successful, but they all have the undesirable effect of increasing the error probability, and therefore they are not immediately applicable to the decoder under consideration. Appendix B gives a description of the techniques along with the results obtained.

* For convenience in terminology, the terms "block" and "message" will be used interchangeably.

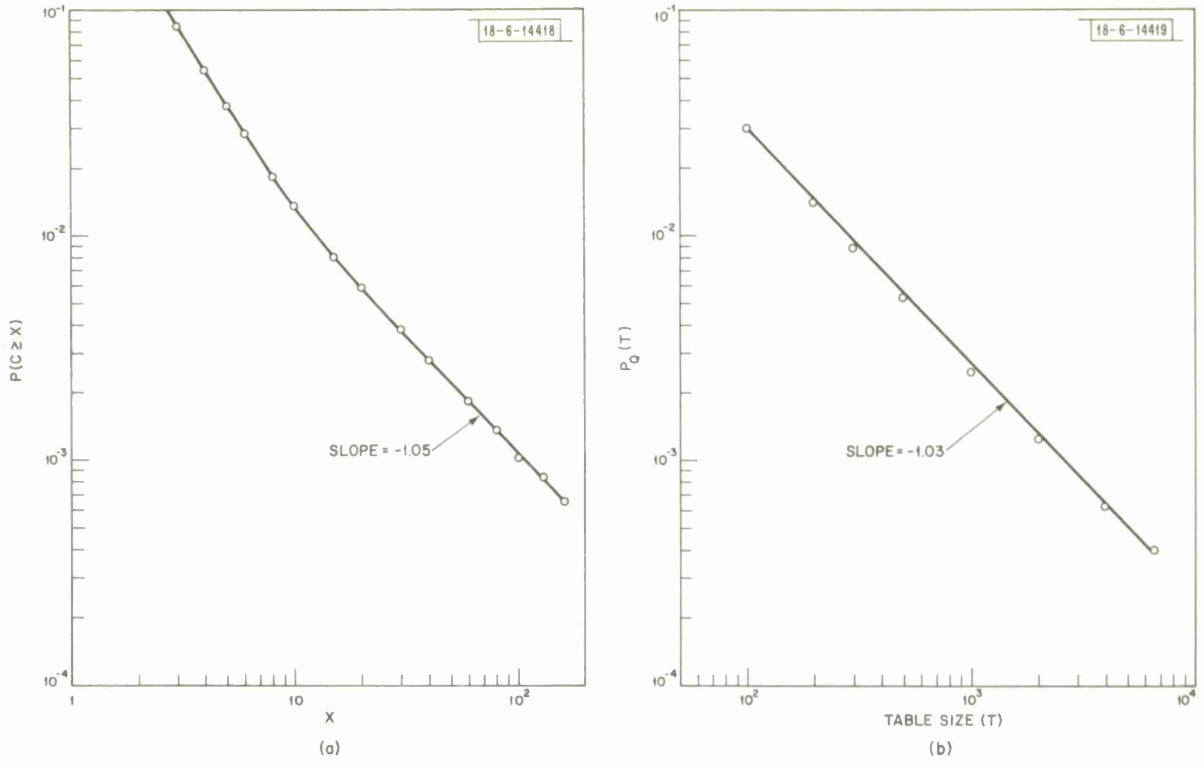


Fig. 6. Decoder performance at $R = 0.99\ R_{comp}$ based upon 2×10^5 simulated messages with N = 100, K = 30, V = 12. (a) Distribution of decoder computations. (b) Decoder quit probability vs table size.

Implicit in each data point of $P_Q(T)$ is a computation cutoff, as outlined in the previous section; the value of the cutoff was chosen to prevent decoder errors that might otherwise be introduced because of the finite memory size (see Section IV-A). Since the computation distribution is for unlimited memory, however, corresponding data from the two curves must be interpreted carefully. Thus, although a failure-to-decode probability of 10^{-3} is associated with $C \approx 100$ and $T \approx 2000$, these two conditions are minimum requirements and are interrelated. If, for example, T is set at 2000 entries, then $C \approx 150$ comp/bit must be permitted to ensure $P_Q < 10^{-3}$. This is because some of the messages that can be decoded in fewer than 100 comp/bit require a table with more than 2000 entries. Conversely, if $T < \sim 2000$, then $P_Q < 10^{-3}$ cannot be achieved, regardless of C, because with high enough probability the correct path has been purged from the table.

The effect of changes in the decoder rate upon the computation distribution and upon $P_Q(T)$ is shown in the curves of Fig. 7. The correspondence between R/R_{comp} and relative SNR for V = 12 and Q = 8 is given in the table accompanying the figure. Each curve (except for $R/R_{comp} = -0.05$ dB which is reproduced from Fig. (6) is based upon 10^4 simulated blocks and is described by a Pareto relationship of the form $BX^{-\beta}$. It should be noted that although the behavior of each curve is eventually dominated by the factor $(X^{-\beta})$ as X increases, for the range of probabilities of interest, the multiplicative factor B is of equal importance. As expected, for operation near R_{comp}, decoder performance is very sensitive to the exact value of the rate. If R decreases by 1 dB, then both the number of computations and the table size required to achieve $P_Q = 10^{-3}$ decreases by approximately an order of magnitude. On the other hand, a slight increase in R above R_{comp} degrades performance to an even greater degree. Operation at $R = R_{comp}$ is therefore the practical choice for the decoder under consideration.

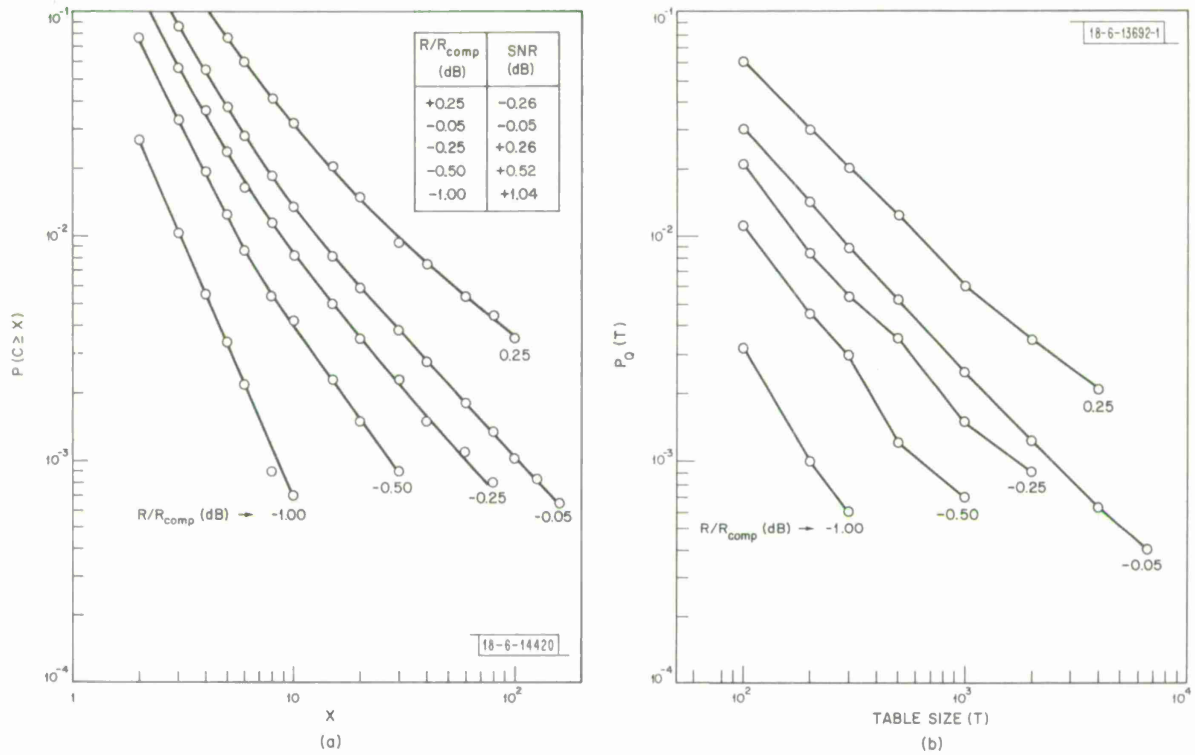


Fig. 7. Decoder performance for several values of R/R_{comp}. The −0.05 dB curves are reproduced from Fig. 6; the other curves are based upon 10^4 simulated messages with parameters as in Fig. 6. The accompanying table gives the relative SNR that corresponds to each value of R/R_{comp} for V = 12, Q = 8. (a) Distribution of decoder computations. (b) Decoder quit probability vs table size.

E. The Effect of Decoder Parameters Upon Performance

Since the performance of a sequential decoder depends primarily upon the ratio R/R_{comp} (Ref. 7), the data presented in Figs. 6 and 7 to a large extent characterize a decoder using the ZJ algorithm. In order to determine the effect of changes in the various decoder parameters upon decoder performance, it is sufficient to determine their effect upon R_{comp} (since $R = 1/V$ is known). Now, parameters such as the message length N and the constraint length K do not affect R_{comp} and so do not greatly influence decoder memory or computation requirements.* The effect of some other parameters, namely, the number of parity networks V, and the number of quantization intervals Q, can be readily determined analytically (cf. Eqs. (4) to (7) and Ref. 8). As a check, a set of simulations with V = 6 and a set with Q = 32 were conducted, with a adjusted in each case to maintain the same R/R_{comp} ratio; in agreement with the prediction, the decoder performance did not change. Finally, the code generator, the metric quantization, and the metric bias are factors which are not amenable to analysis; simulations were necessary in order to ascertain their quantitative effect on decoder performance.

*The constraint length determines the error performance, which is discussed in the next section. It is possible that, because N is on the order of three constraint lengths, the results obtained with this decoder are somewhat atypical; however, a limited number of simulations with N = 1000 gave results roughly comparable to those with N = 100.

The code generator is a KV-bit vector that specifies the parity network connections between the V modulo-2 adders and the K-stage shift register. Let c_i denote the K-bit connection vector for the i^{th} parity network ($i = 1, 2, \ldots, V$). The $\{c_i\}$ used in the simulations of Figs. 6 and 7 were (in octal notation):

$c_1 = 7630633135$ $\quad$ $c_7 = 5517570324$

$c_2 = 7255122155$ $\quad$ $c_8 = 5216661277$

$c_3 = 7543155131$ $\quad$ $c_9 = 5742601204$

$c_4 = 6044024066$ $\quad$ $c_{10} = 4516110527$

$c_5 = 6422351171$ $\quad$ $c_{11} = 4753030345$

$c_6 = 6231240635$ $\quad$ $c_{12} = 4162000625$

The first 5 bits of each c_i were chosen (manually) to maximize the minimum Hamming distance between transmitted codewords;[9] the remaining 25 bits of each c_i were chosen at random. Now, because we are dealing with a message length only a few times greater than the constraint length, and because of symmetry in the encoding process, it was conjectured that performance would improve if the final bits of each c_i were selected by a procedure similar to that used in selecting the initial bits. For example, the first bit of each c_i is 1 (each parity network is connected to the first shift register position), so that the V encoded bits on the two branches stemming from any tree node are logical complements (and the associated Hamming distance is thus V, the maximum possible). In particular, the first information bit generates either V 0-bits or V 1-bits since the shift register is initially filled with 0's. But by the same token, when the final tail bit is shifted in, the shift register contains all 0's except for the last information bit, which occupies the K^{th} shift register position. It therefore seems desirable to set the K^{th} bit of each c_i to 1. Simulations were performed with the last 5 bits of each c_i selected to maximize the minimum Hamming distance of the codewords that would be generated if the information sequence were fed into the K^{th} position rather than the first position. The performance of the decoder with these $\{c_i\}$ was comparable to that of the original, indicating that the original conjecture was incorrect.

The fact that metric values cannot be kept to full precision also influences decoder performance. The degree of this quantization is determined by the range of the metric increments, as specified in Eq. (9a). Maximum precision is achieved, of course, by allowing a large range. But if the range is too large, the metric along a complete path (viz., the sum of $V(N + K - 1) \sim 10^3$ increments) may be greater than 2^{15} and consequently more than one word would be required to hold the metric value on a 16-bit computer. This would increase the memory requirements and the computation time and is therefore undesirable. On the other hand, if the range is too small, many nodes will have similar metric values, and decoder performance will degrade because the correct node will be indistinguishable from numerous incorrect nodes. For purposes of comparison, let us deal with the number of bits required to specify the range of metric increments, k-bit quantization meaning that the difference between the largest and the smallest increments is $(2^k - 1)$. The data of Figs. 6 and 7 are based upon 8-bit increments. Simulations were performed with 10-bit increments but no improvement in decoder performance was realized, and 7-bit increments did not cause any degradation. When 4-bit increments were used, however, there was a marked increase in P_Q. The conclusion is that 7- or 8-bit increments provide performance equivalent to that without metric quantization, and that the use of this range of

increments ensures that, with very high probability, each node metric occupies a single 16-bit word; 8-bit increments were used for all other simulations discussed in this report.

As mentioned in the previous section, the metric bias U also affects the operation of the decoder. For each curve of Fig. 7, the bias was set equal to the rate $U = R = 1/V = 0.083$. A number of these simulations were repeated with $U \neq R$, but the choice $U = R$ gave the best results. Values of U very close to R did not alter performance significantly, but values that differed from R by more than ~10 percent caused noticeable degradation in the computation distribution and in $P_Q(T)$.

IV. ERROR PERFORMANCE

One of the features of sequential decoding is the low probability of error that can be economically achieved at low SNR's. Furthermore, the reliability of decoded data can be substantially increased with only a small increase in decoder complexity. In this section we discuss the error performance of the ZJ algorithm and consider several innovations for improving this performance.

The probability that a bit is incorrectly decoded is bounded above by an expression of the form (Ref. 6) $B2^{-\beta K}$, where K is the constraint length and where B and β depend primarily upon the channel. With N information bits in a block, the probability P_E of one or more errors in the block is then certainly less than the union bound

$$P_E < NB2^{-\beta K} \quad . \tag{10}$$

Thus, an increase in K reduces P_E by an exponential factor; moreover, an increase in K (within certain ranges) does not add to decoder complexity at all, when implementation is on a digital computer. However, since $K - 1$ tail bits are transmitted with each message, and since the message blocks are short, any increase in K entails a material increase in transmitter power (or in transmission time). For example, with N = 100, if K changes from 30 to 33, the transmitter power necessary to maintain the same SNR increases by 0.1 dB. The tail must therefore be kept as small as possible. In Eq. (10), $\beta = 1$ for operation at $R = R_{comp}$. For convenience in terminology, let us henceforth define an SNR of 0 dB to correspond to this operating point.* For SNR's greater than this nominal 0 dB, P_E decreases, but if SNR < 0 dB, β decreases and P_E increases. Now, if an accurate measurement of the SNR over a block could be obtained, then those blocks with SNR < 0 could be ignored. That is, rather than increase the likelihood of a decoding error, no output would be provided if SNR < 0. This procedure would meet the requirements given in Section I because there is a specification on P_Q only at (or above) 0 dB. The procedure is, of course, impractical because the SNR cannot always be determined to sufficient precision, and it is also objectionable because there would be no chance of decoding a message if the SNR decreased slightly. Thus, it is both necessary and desirable to have a scheme that (a) provides the specified high reliability regardless of SNR, (b) gives a reasonable probability of decoding if the SNR is slightly below nominal, and (c) does not cost much in terms of transmitter power (or, equivalently, number of transmitted bits) or in terms of decoder performance under normal conditions. The scheme developed in this section provides all these features.

---

* In other words, the SNR used here is <u>not</u> numerically equal to the ratio E_b/N_o (or E_s/N_o), but differs from that ratio by a constant factor. For example, an SNR of −1 dB means that E_b/N_o (or E_s/N_o) is 1 dB below the value that is necessary to achieve $R/R_{comp} = 1.0$. The equations of Section II-B can be used to calculate E_b/N_o.

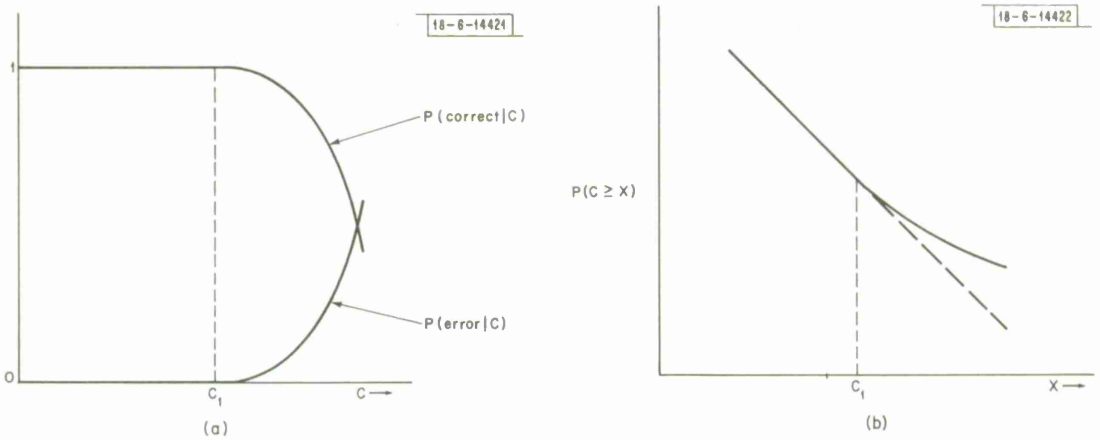


Fig. 8. Qualitative behavior of the error performance of the ZJ algorithm operating with a finite memory. (a) Increase in the likelihood of an error as the number of decoder computations increases. (b) Shallow slope of the computation distribution as the number of computations increases (log-log coordinates).

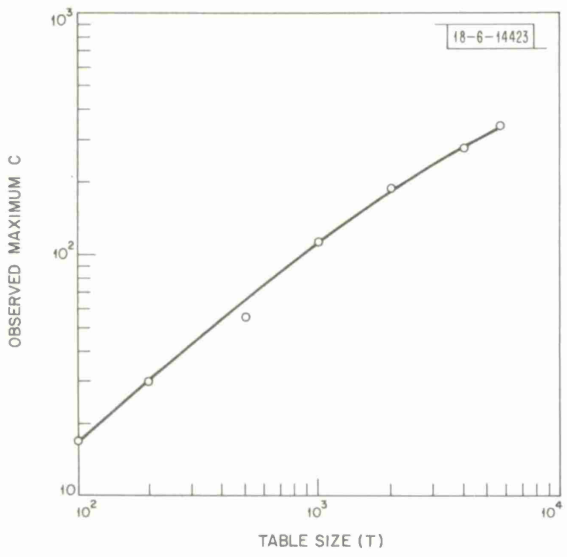


Fig. 9. The maximum number of computations observed during the simulations for Fig. 6(b) (all messages decoded correctly) as a function of table size at $R/R_{comp} = 0.99$. This curve may be used as a guide in selecting C_{max}.

A. The Effect of Finite Memory Upon P_E

The bound in Eq. (10) was originally developed for a sequential decoder using the Fano algorithm, but it also applies to one using the ZJ algorithm provided that unlimited memory is available.[5] As mentioned in Section III-C, however, the ZJ algorithm operating with limited memory possesses an additional source of error because, if the correct path is ever purged from the memory table, it can never be recovered, and an error will occur if decoding is permitted to continue until the end of the decoding tree is reached. We now demonstrate that the errors caused by the finite memory can be eliminated with a computation cutoff that restricts the number of computations performed by the decoder.

Let C_{max} denote the average number of computations per bit allowed for decoding; i.e., the decoder quits at $C = C_{max}$ or, equivalently, when the total number of computations equals NC_{max}. Consider the extreme case of ZJ algorithm operating with a table size of $T = 2$ entries, so that at any time only the two largest metrics can be saved. Clearly, if such a decoder ever recalls a metric from memory (i.e., if it ever has to back up in the decoding tree), there is a high probability that an error will be made. With $T = 2$, then, the proper choice is $C_{max} = 1.00$, and a message is decoded only if the minimum possible number of computations is required. Now as T increases, the allowed number of computations increases, but corresponding to each value of T there is a C_{max} beyond which the error probability of a decoding message increases greatly. This principle is illustrated qualitatively in Fig. 8(a). For a fixed T, the figure shows that if a message is decoded, then the probability that there is an error in the message increases significantly for C greater than some value, say C_1. Quantitative information on this phenomenon may be extracted from the computation distribution $P(C \geq X)$, as sketched in Fig. 8(b). The distribution exhibits the expected Pareto behavior for $C < C_1$, but for $C > C_1$ tends to decrease at a slower rate.* The computation limit C_{max} is chosen somewhat before the knee of the curve. Because of the Pareto nature of the curve, the exact choice of C_{max} affects neither P_Q nor P_E critically. Figure 9 shows the maximum number of computations as a function of T that were observed during the simulations used to obtain Fig. 6(b). These data can be used as a guide in selecting C_{max} for a particular T. In order to obtain more specific data, simulations were performed at $R = R_{comp}$ with K small enough that decoding errors could be observed. Figure 10 shows the results for $K = 15$, 17, and 19 with $T = 2048$. The curve of P_Q is equivalent to the computation distribution function; i.e., if the decoding is stopped after C_{max} computations, then the quit probability is $P_Q|_{C_{max}} = P(C > C_{max})$. As expected, the error probability P_E increases as more computations are allowed. The sum $(P_Q + P_E)$ is, however, flat beyond $C_{max} \sim 180$, indicating that beyond this value the decoded message is likely to be in error. In other words, nothing is lost if decoding is terminated when $C = 180$. Note also that, regardless of the value of K, the curves have roughly the same shape and that the break points occur at about the same value of C_{max}. The fourth curve in each set, P'_E, is an empirical lower bound on the error probability that would obtain with infinite memory. The points on this curve were generated by noting those error messages which satisfied either (or both) of the following conditions:

* For still larger values of C, the distribution drops sharply if $P_Q \sim P_E$ because of the sharp rise in P_E.

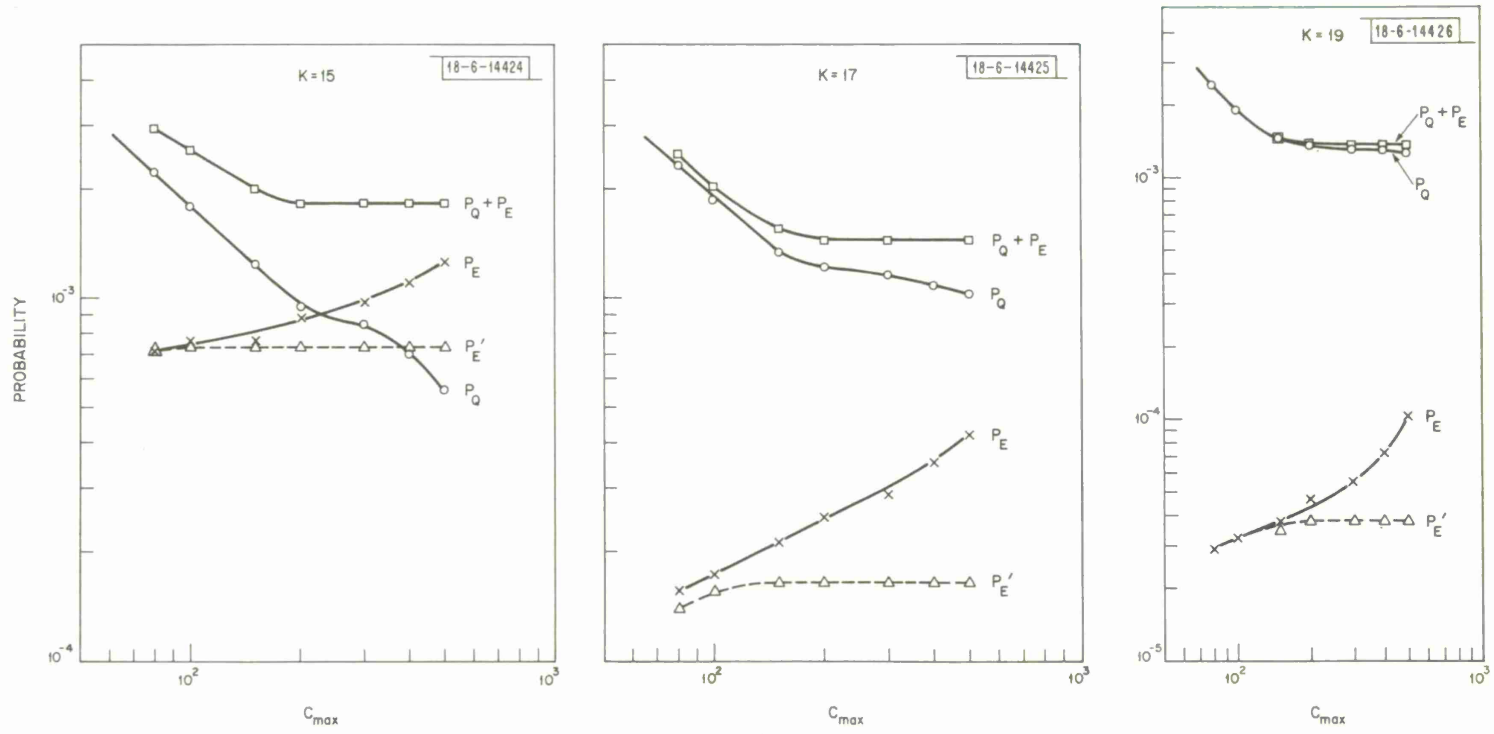


Fig. 10. Empirical curves showing the quit probability and the error probability as a function of the computation cutoff for several values of K at $R/R_{comp} = 1.00$, T = 2048. See text for a description of P_E'.

(1) The correct path was not purged from memory. In this case, the correct path was available throughout the decoding process, but the incorrect path was preferred.

(2) At any point in the decoding process, the largest metric – namely, the one to be pursued by the decoder – was equal to or greater than the largest metric that had been discarded. In this case, the decoder would never have examined any of the discarded entries, even if they had been retained in memory.

Any error that occurs when either of these conditions is satisfied would also occur if unlimited memory were available. However, since a message that could not be decoded with limited memory might become an error message if memory were infinite, P'_E represents a bound and not an equality. The data of Fig. 10 show that if $C_{max} < \sim 180$, then the errors due to the finite memory size are eliminated.

Thus, in addition to limiting the time spent decoding a message, the restriction $C \leqslant C_{max}$ limits errors to those that would be made with the ZJ algorithm operating with infinite memory. Henceforward, the use of a computational cutoff is implicitly assumed.

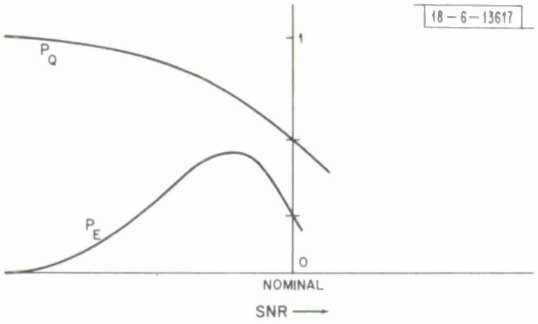


Fig. 11. Qualitative behavior of the quit probability and the error probability for SNR's lower than the design value of 0 dB.

The computational cutoff, in fact, serves a third purpose: it helps reduce P_E as the SNR decreases because it causes the decoder to quit on an increasing fraction of the messages. Figure 11 illustrates this effect. If the SNR is very low, then $P(C > C_{max}) = P_Q \to 1$, and a fortiori $P_E \to 0$. (P_E is computed over all transmitted messages, not just over decoded messages.) We find empirically, however, that as the SNR decreases from 0 dB, the initial increase in P_Q is not sufficient to compensate for the increase in P_E described by Eq. (10). The problem, then, is to ensure that the peak P_E does not exceed specifications.

B. Threshold on Final Metric Value

As mentioned earlier, the simple expedient of increasing K sufficiently is not satisfactory because of the associated penalty in transmitter power needed for the tail. The peak P_E can be reduced to some extent, without incurring any penalty, by applying a threshold Θ on the final decoder metric, rejecting a message if the final value of its metric, M, falls below Θ. Since a message that fails the threshold becomes a quit, the threshold must be chosen to have a negligible effect on P_Q at 0 dB. Figure 12 depicts the probability density function p_{co} for the final metrics of all correct paths at SNR = 0 dB and the corresponding densities for all correct and incorrect paths at some SNR < 0. When conditioned by $C \leqslant C_{max}$, each of the density functions shifts to the right. Because Θ must be far out on the tail of p_{co}, the fraction of errors eliminated by the threshold, while useful, will not be very close to unity. Figure 13 shows experimental results

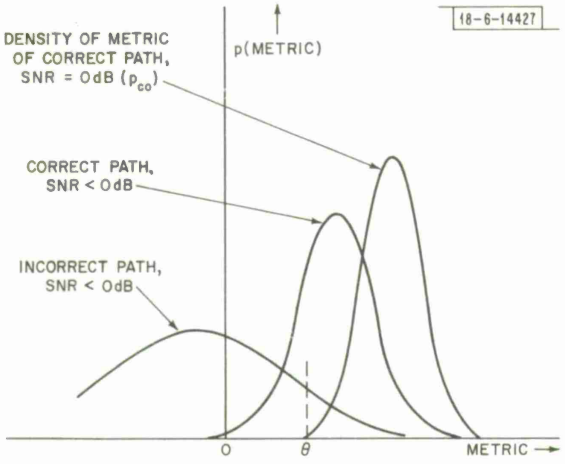


Fig. 12. Probability density functions of several metric values. The threshold θ must be chosen far on the tail of the 0-dB correct-metric density, and therefore a significant fraction of incorrect paths will exceed the threshold at lower SNR's.

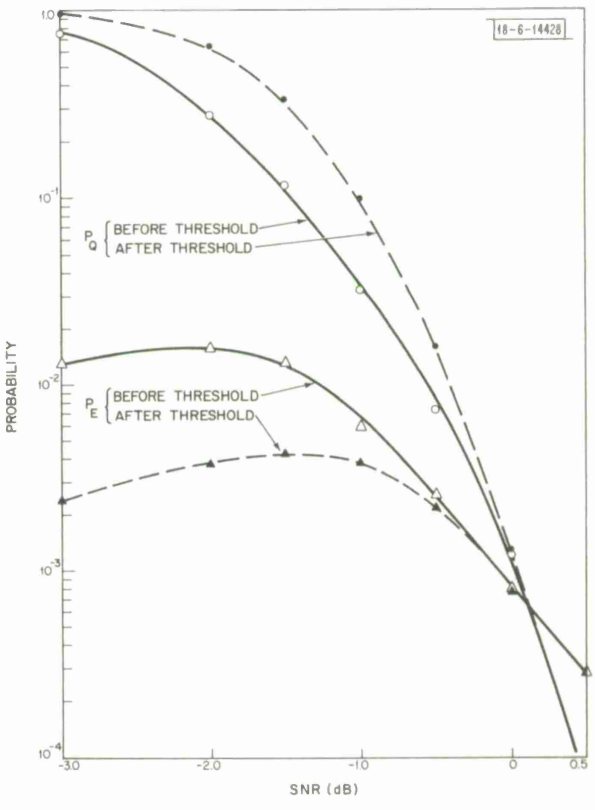


Fig. 13. Quantitative curves of the quit probability and the error probability (with K = 15, T = 2048, C_{max} = 150) as a function of SNR both before and after applying a metric threshold (cf. Fig. 11). The use of the threshold reduces the peak P_E.

for P_Q and P_E as a function of SNR both before and after applying a threshold.* The data are for a decoder with K = 15 and C_{max} = 150. The density p_{co} is Gaussian, and Θ was chosen at the (-3.29σ)-point, i.e., such that $P(M_c < \Theta) = 0.5 \times 10^{-3}$, where M_c denotes the correct path metric. Actually, the probability of a threshold failure at 0 dB is significantly less than this value because, as noted above, a large fraction of those messages with $M_c < \Theta$ require excessive computations and become decoder quits. As seen from Fig. 13, the threshold does indeed have a negligible effect on P_Q at 0 dB, but the threshold lowers the peak P_E by a factor of 4.

C. Use of a Long Shift Register

In order to achieve a further reduction in P_E, several other approaches were considered. The first two approaches that were tried – viz. (1) a running metric threshold, and (2) forward and backward decoding – yielded some improvement in P_E. However, they both possess significant drawbacks (primarily, an increase in the required table size), and therefore they are not suitable for the decoder under consideration. An explanation of these techniques is given in Appendix C.

A superior approach, proposed by M. Hellman[10] uses a long shift register and achieves a substantial reduction in P_E with no increase in P_Q and with negligible increase in decoding time. Consider a convolutional encoder/sequential decoder having a shift register of length S equal to the total number of bits transmitted. Thus, if there are N information bits and L tail bits, S = N + L. Since the i^{th} information bit remains in the shift register for S + 1 − i shifts, the error probability associated with this bit is $B2^{-(S+1-i)}$ for operation at 0 dB, and P_E is therefore bounded by

$$P_E < B \sum_{i=1}^{N} 2^{-(S+1-i)} = B2^{-L} \sum_{j=1}^{N} 2^{-j} \qquad (11)$$

$$< B2^{-L} \quad .$$

This is significantly lower than the corresponding union bound given by Eq. (10) for a conventional decoder with K = L + 1, and the improvement will also prevail at lower SNR's. In practice, the shift register length need not be N + L because the same P_E results if merely $S \gg L$; for L ~ 30, S = 64 (four words on a 16-bit computer) or even S = 48 suffices. An interesting and useful feature of this type of decoder is that, with N = 100, the first ~60 bits are very reliable since the error probability associated with this part of the message is

$$< \sim 60\ B2^{-\beta S} = 60\ B2^{-64\beta} \qquad \text{(if S = 64)} \quad .$$

Therefore, the first part of any decoded message, regardless of the value of its final metric, can be accepted with confidence.

---

*The actual signal and noise levels that were used to obtain a given SNR were equivalent to those that would result if an AGC preceded the decoder, where the AGC maintained the total energy (signal + noise) constant. For SNR's near 0 dB, a change in the SNR is reflected primarily in a change in signal level.

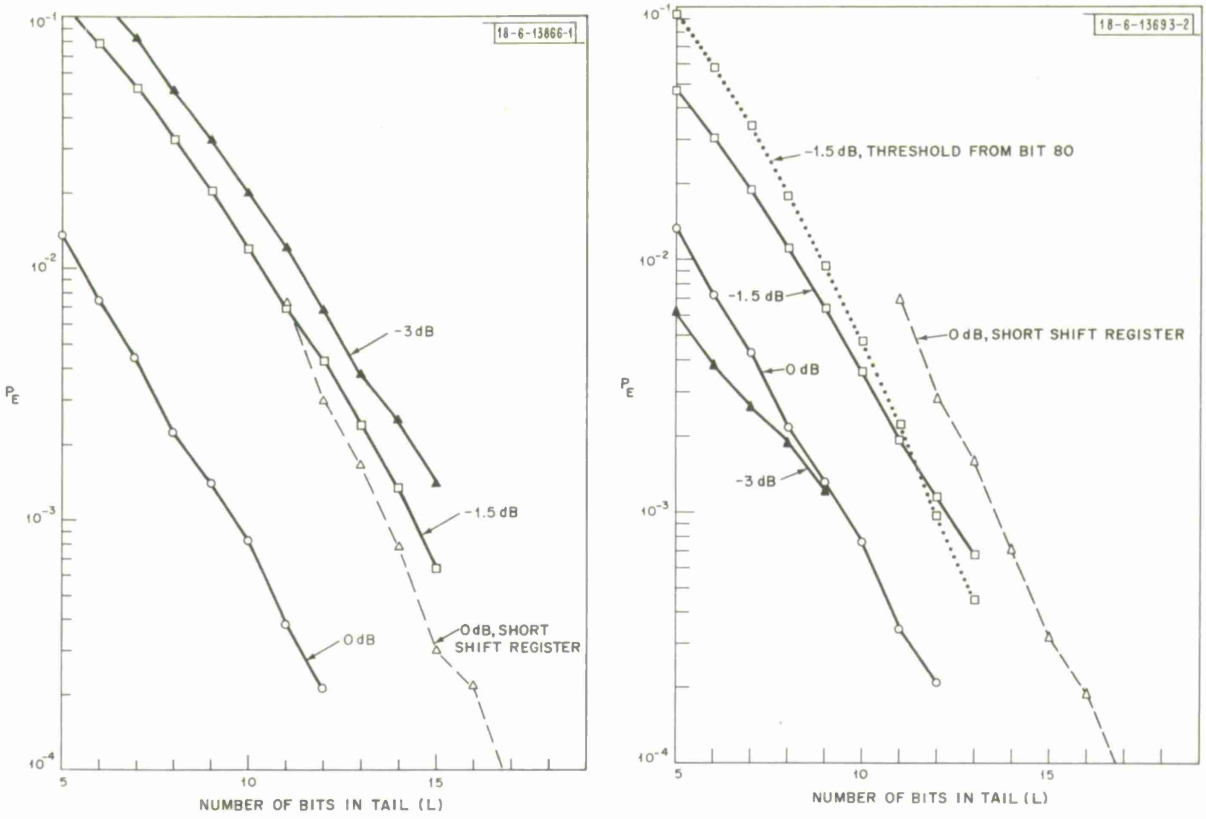


Fig. 14. Error probability as a function of the number of tail bits for several SNR's with a long shift register, and at 0 dB with a short (i.e., length L + 1) shift register (T = 2048, C_{max} = 150). The SNR's do not include the penalty associated with the tail. (a) Before applying metric threshold. (b) After applying metric threshold.

Some quantitative results for a decoder with a long shift register are shown in Fig. 14. The first set of curves [Fig. 14(a)] gives P_E as a function of L for several SNR's. Each curve displays the predicted exponential behavior. For clarity, a curve at −4 dB has been omitted from the figure, but this curve would lie below the −3 dB curve. Thus, for a particular value of L in the range shown, a plot of P_E as a function of SNR exhibits the behavior depicted in Fig. 11, with the peak P_E occurring near −3 dB. (The curves of Fig. 13 are not for a decoder with a long shift register.) The dashed curve of Fig. 14(a) is P_E vs L for a conventional sequential decoder (with S = K = L + 1) at 0 dB. At this SNR, it is seen that the long-shift-register decoder yields more than a factor of 15 improvement in P_E over the conventional decoder. It is important to note that the SNR used here does not take into account the additional power associated with the tail; this power, which of course increases with increasing L, can be computed after the required number of tail bits is determined.

Figure 14(b) gives curves corresponding to those of Fig. 14(a) after the metric threshold Θ is applied. A comparison of the two sets of curves shows that at 0 dB the effect of the threshold is negligible, but as the SNR decreases, an increasing fraction of the errors is eliminated. In fact, the −3 dB curve now lies below the 0 dB curve. Again, curves at −1 dB and −2 dB have been omitted, but both would lie below the −1.5 dB curve, indicating that the post-threshold peak of P_E occurs near −1.5 dB.

The dotted curve shown in Fig. 14(b) is the result of applying a threshold (at −1.5 dB) on the metric accumulated from bit 80 rather than from bit 0. This procedure becomes advantageous as the number of tail bits increases because of the high reliability of the initial decoded bits with a long shift register. Thus, if there is a decoding error, the branch metrics from depth N through depth N + L are "error metrics" (with a mean value that is large and negative), whereas the metrics from depth 0 through depth ~60 are almost certainly "correct metrics." If the change in metric only over the tail bits is measured, the small number of samples contributing to the measurement result in a relatively large variance. Since the threshold must be chosen to have only a small effect on P_Q at 0 dB, this large variance would dictate a threshold that would be too low to reject an appreciable fraction of the error messages at low SNR's. It was found empirically that at −1.5 dB (corresponding to the peak P_E) with L ~14, best results are obtained by using the change in metric between depth ~80 and depth N + L ~114. The threshold applied over the entire message may be considered a test of the SNR, whereas the threshold applied over the final bits may be considered a measure of the error probability.*

It is interesting to note that, for the long-shift-register decoder, the data for an entire P_E vs L curve can be obtained easily by the following technique. For each simulated message, the decoder is initially set for L = 5, the smallest value of interest. If decoding is completed before the computation cutoff, a record is made of whether or not there was an error in the message. Then L is incremented by one, and decoding is allowed to continue. This procedure is valid because the state of the decoder is identical to that of a decoder with total tree depth 106 that has just examined the best path at depth 105. The process continues until L attains the maximum value of interest. If for L = L' the decoder reaches computation cutoff, then a quit is entered for all $L \geqslant L'$. Thus, for each simulated message, information on all values of L is obtained with minimal extra effort. (With a conventional decoder each message must be re-run for each value of constraint length K. At first it was assumed that if a message was decoded correctly for $K = K_1$, then for $K > K_1$ the message would again be decoded correctly. However, the assumption proved false when in fact it was observed that many correctly decoded messages at $K = K_1$ became errors at $K_1 + 1$, even though the first K_1 bits of the parity network connection vectors were unchanged. Thus, each point of the short-shift-register data required as much effort as an entire curve with the long shift register.)

D. Obtaining Data for Very Low P_E; Decoding into the Tail

If the exponential curves of Fig. 14(b) are extrapolated to $P_E = 10^{-4}$ and to $P_E = 10^{-7}$, the values of L shown in Fig. 15 are obtained. (The short shift register, −1.5 dB curve is not given in Fig. 14.) The extrapolated values at −1.5 dB dictate the requirements on L, since this SNR corresponds to the worst-case P_E. Actually, the values in the figure are conservative because the threshold test becomes more effective as the total number of tail bits increases; that is, the fraction of error messages that passes the threshold decreases as L increases. The data indicate that, to obtain $P_E < 10^{-4}$, the long-shift-register decoder required about three less tail

*This fact suggests the use of two thresholds — one on the metric from depth 0 and one on the metric from depth ~80 — with the requirement that both thresholds be exceeded in order for a message to be accepted. An experiment was conducted with each threshold set so that 0.3×10^{-3} of all correct metrics at 0 dB would be discarded. (With a single threshold, the value of 0.5×10^{-3} was used.) The error rejection improved slightly, but conclusive results could not be obtained with the relatively small values of L that were necessary in order to observe a statistically significant number of errors.

18-6-14429

	SNR (dB)	NUMBER OF TAIL BITS REQUIRED	
		$P_E < 10^{-4}$	$P_E < 10^{-7}$
LONG SHIFT REGISTER	0	14	25
	−1.5 (metric from bit 0)	17	30
	−1.5 (metric from bit 80)	16	25
SHORT SHIFT REGISTER	0	17	27
	−1.5	20	31

Fig. 15. Number of tail bits required to achieve indicated error probabilities at 0 dB and at −1.5 dB (which corresponds to the peak of the P_E vs SNR curve). The values were obtained by extrapolating the data of Fig. 14(b).

bits than the conventional decoder. However, since the long-shift-register curves have a somewhat shallower slope, only one less tail bit is needed for $P_E < 10^{-7}$. However, if a threshold on the metric from bit 80 is used, only 25 tail bits are needed – fewer than would be required with a short shift register operating at 0 dB!

Unfortunately, however, there is no convenient way to determine the accuracy of these extrapolated data. Analysis is formidable because of the conditional probabilities involving computation cutoff and metric threshold, and simulations are impossible because of the inordinate computer time necessary to obtain data on such low probability events.* A margin of safety can be inserted into the extrapolated data by augmenting several bits to the indicated values of L but, as mentioned earlier, three tail bits represent an additional 0.1 dB transmitter power, so this solution is to be avoided. Fortunately, with the long-shift-register decoder there is a simple way to overcome this problem.[10] The decoder treats the first L_1 tail bits as if they were information bits (i.e., the decoding tree has two branches per node up to depth $N + L_1$), but after decoding is complete, a check is made to see that these known bits are indeed correct. It can be shown that if an error is made on any of the information bits $(N - S + L_1 + 1), (N - S + L_1 + 2), \ldots, (N + L_1)$ – viz., those bits that are still in the shift register when bit $N + L_1$ is shifted in – then the probability that the L_1 tail bits are decoded correctly is 2^{-L_1}. Furthermore, the initial information bits $1, 2, \ldots, N - S + L_1$ are known to be reliable because a long shift register is being used. The technique achieves good results because it halves P_E for each bit in L_1, whereas each extra tail bit reduces P_E by only $2^{-\beta} > 2^{-1}$ if SNR < 0. As an example, L = 17 gives $P_E = 10^{-4}$ (cf. Fig. 15), so $L_1 = 10$ ensures that $P_E = 2^{-L_1}(10^{-4}) < 10^{-7}$, and a total of 17 + 10 = 27 tail bits is needed. Alternatively, with $L_1 = 13$ and 13 additional tail bits, $P_E < [(1/8) \times 10^{-3}]\,[8 \times 10^{-4}] = 10^{-7}$, with no extrapolation of the data. According to the curves of Fig. 14(b), this value of

* For example, a data point at $P_E = 10^{-5}$ requires $\sim 20 \times 10^5$ simulated messages so even if the average time to process a message were 1 second, approximately 500 hours of computer time would be required. Moreover, at −1.5 dB, where $P_Q > 0.1$ and where the expected number of computations for a decoded message is high, the average decoding time is considerably more than 1 second.

26 total tail bits is the minimum necessary to ensure $P_E < 10^{-7}$. Note that L_1 should be significantly less than L so that the quit probability at the nominal 0 dB is not appreciably affected. Since the probability of an error occurring in one or more of the first L_1 tail bits (at 0 dB) is on the order of $2^{-(L-L_1)} + 2^{-(L-L_1+1)} + \ldots + 2^{-L} \approx 2^{-(L-L_1-1)}$, $(L - L_1)$ must be chosen such that $2^{-(L-L_1-1)} \ll 10^{-3}$, namely $(L - L_1) \gtrsim 13$.

V. IMPLEMENTATION OF THE ZIGANGIROV-JELINEK ALGORITHM

The results presented in Section III show that a small, general-purpose digital computer has adequate memory to perform sequential decoding using the ZJ algorithm: a table containing on the order of 2000 entries achieves $P_Q = 10^{-3}$ at $R = R_{comp}$. In this section we compare several techniques for efficient implementation of the algorithm. The principal conclusion is that a typical computer has adequate speed for the decoding task.

A. Details of a Practical ZJ Algorithm

A flow chart for the ZJ algorithm is given in Fig. 16. The flow chart differs from the basic five-step algorithm given in Section III because some time-saving features are included. Most of the steps are self-explanatory, but the following points deserve elucidation:

(1) Initialization involves setting up the memory tables and setting the CURRENT entry to correspond to the start of the decoding tree.

(2) Whenever Step A of Fig. 16 is performed, CURRENT represents the node having the largest metric, i.e., the node on the path that will be extended next; BEST represents the node having the second-largest metric, i.e., the table entry that has the largest metric; and WORST represents the node having the smallest metric, i.e., the table entry that, if necessary, will be discarded.*

(3) When the current node is extended, two metrics, M_1 and M_2, are produced (unless the current node is in the tail), with M_1 designating the larger of the two. If M_1 is greater than the best metric, then M_1 need never actually be placed on the table since it corresponds to the path that will be extended (Step B).

(4) No bookkeeping need be done on WORST until the table becomes full (Step E).

(5) Once the table is filled, one entry must be discarded for each computation (unless the current node is in the tail). However, if the table is full and if M_2 is less than the worst metric, then M_2 is discarded immediately (Step F). This test saves considerable time when a noisy message is processed.

(6) Step J represents the test for computation cutoff.

B. Data Structures for the ZJ Algorithm

When the ZJ algorithm was initially considered for implementation, no information was available on the quantitative effect of the table size T on the quit probability P_Q. It was thus

* It is not obvious that the smallest metric is necessarily the "worst" metric – that is, the metric least likely to be on the correct path. A number of simulations were conducted in which the discarded node was the one with the smallest value of the quantity $(M + d\Delta)$, where M is the metric, d is the depth, and Δ is a parameter. The value $\Delta = 0$ thus corresponds to eliminating the smallest metric; $\Delta < 0$ implies that nodes farther along in the tree are preferred; $\Delta > 0$ implies that nodes farther in the tree tend to be discarded. The conclusion from these experiments was that at $R = R_{comp}$ there is no particular advantage in considering the worst node to be other than the one with the smallest metric.

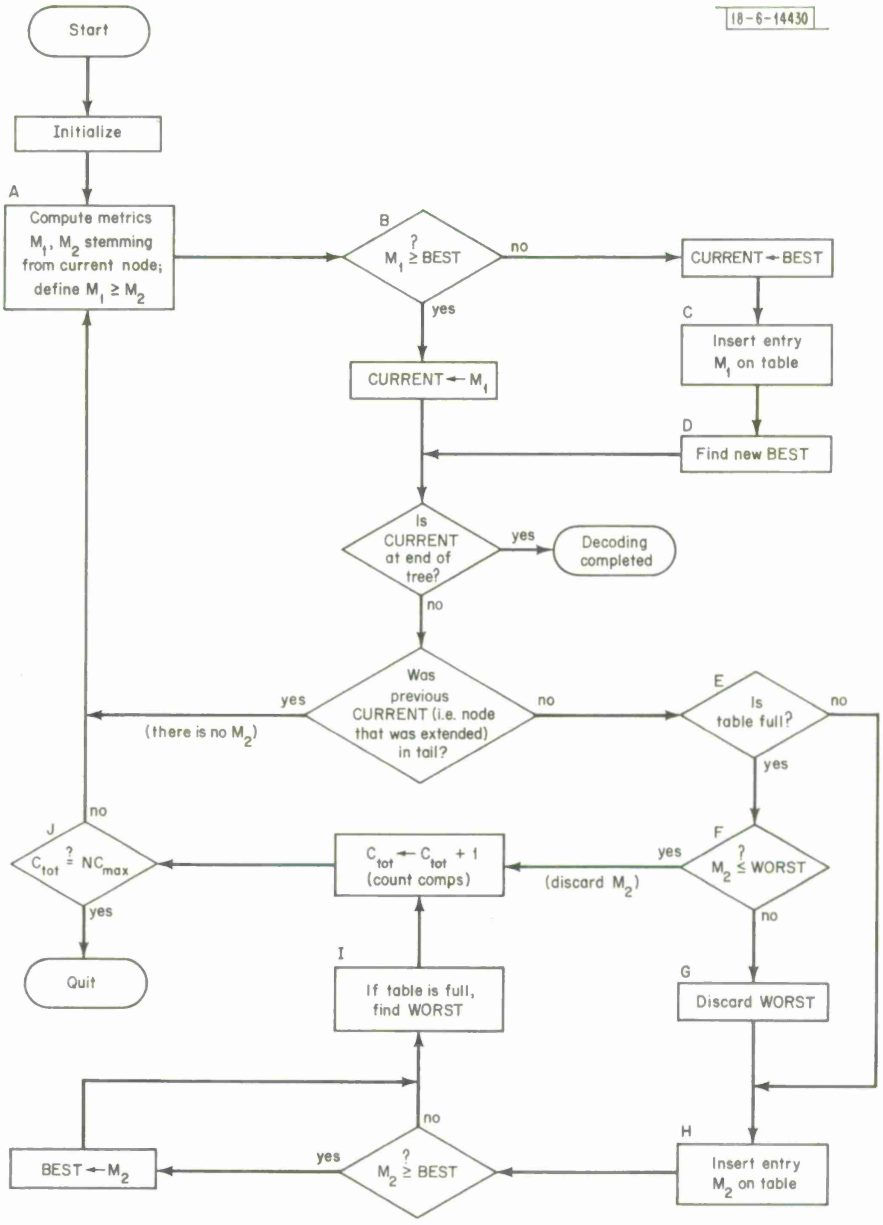


Fig. 16. Detailed flow chart of the ZJ algorithm.

difficult to determine the most efficient data structure, and it was impossible to predict which of the operations of Fig. 16 would consume the most time. The most expeditious approach, therefore, was to employ the simplest implementation, then establish what value of T is required, and finally determine an implementation for this value of T that would reduce the decoding time.

1. Unordered Table with Linear Search

For the reasons described above, the initial implementation kept the temporarily discarded nodes on an unordered table. This data structure requires the least extra memory, is easiest to program, and is conceptually the simplest. In fact, this data structure would be satisfactory if the required T were small (~300 entries). When it was determined that T ~ 2000 is necessary, however, it became clear that the time spent searching the unordered table for the best and worst metrics (Steps D and I) was excessive. This is shown by the data in the first column of Fig. 17, which gives the average number of memory cycles per decoder computation, as implemented on a Varian 620/i computer.* A decoder computation includes all the operations in the

18-6-14431

	LINEAR TABLE	QUADRATIC TABLE	CUBIC TABLE	TABLE-LIST STRUCTURE (estimate)
PARITY & METRIC (t_{PM})	920	920	920	920
DATA STRUCTURE (t_{DS})	14400	1040	520	150
MISCELLANEOUS (t_{MISC})	100	100	100	100
TOTAL (t_{COMP})	15420	2060	1540	1170

Fig. 17. Average number of memory cycles per computation for several implementations for a message that cannot be decoded within the computational cutoff. The first three columns of data were obtained from actual implementations, whereas the last column is an estimate. All values are for a Varian 620/i and are for T = 2048.

loop that returns to Step A of Fig. 16. Since the branches taken within this loop vary from computation to computation, only the average time is meaningful. The values shown are an average for a number of messages that, because of excessive noise, could not be decoded after 15,000 computations (N = 100, and C_{max} = 150), with operation at R = R_{comp} = 1/12, and with T = 2048, the shift-register length S = 64 bits, and the tail L = 29 bits. (Computations in the tail are not included in the total number of computations.) Note that it is the messages that cannot be decoded that limit the real-time decoder performance, and therefore it is these messages which must be considered when speed and memory requirements are determined. The total time t_{COMP} is broken into three categories:

$$t_{COMP} = t_{PM} + t_{DS} + t_{MISC}$$

* The Varian 620/i is a 16-bit computer having one accumulator, two index registers, and a memory cycle time of 1.8 microseconds.

in which t_{PM} is the time required to compute the V parity bits and the metric for both of the branches from a node; t_{DS} is the time required to process the metrics and to insert them into the data structure; and t_{MISC} is the time spent performing other operations such as updating the shift register and counting computations. For the original implementation, then, t_{DS} occupies more than 90 percent of the decoding time, whereas less than 7 percent of the time is spent performing the basic decoder functions included in t_{PM}.*

Since the time t_{DS} depends heavily upon the data structure, an implementation more efficient than the unordered table must be found. Now, the unordered table requires excessive time because a linear search of the entire table must be initiated each time the largest or smallest metric must be found. The time associated with this type of search is given by

$$t(\text{linear search}) \approx (T)\,(t_c) \quad ,$$

where t_c is the time required for the instruction loop which compares two metric values and indexes down the table. The overhead instructions associated with setting up the loop are ignored for the present calculations. Depending upon the instruction repertoire available, t_c will generally lie in the approximate range $5 \leqslant t_c \leqslant 10$ computer cycles; for the Varian 620/i, which has no compare instructions, $t_c \approx 9$ cycles. Since $T \sim 2000$, the linear search requires on the order of 18,000 cycles, and this search must be carried out each time Step D or Step I is performed. The data in Fig. 17 reflect the fact that these operations are not performed for each decoder computation. In fact, a limited number of simulations indicated that for worst-case messages (i.e., those that could not be decoded before computation cutoff), the probability of having to find a new BEST is ~0.65, and, when the table is full, the probability of having to discard the worst table entry (rather than M_2) is only ~0.20. These data show that during a noisy message, with the decoder spending most of its time examining incorrect nodes, (a) an incorrect path is pursued, on the average, for $\sim(1/0.65) \approx 1.5$ nodes, and (b) the range of metric values in memory is so small that the lesser of the two metrics generated at a node is, with probability ~0.80, less than all other table metrics. Even though a table search is executed only $\sim 0.85 = 0.65 + 0.20$ times for each computation, the computing time associated with a linear search of an unordered table is too great.

2. Ordered List and Ordered Table

In order to reduce the search time, alternative data structures were considered. An ordered list[11] appeared advantageous because it was expected that, when the largest metric was incremented during a decoder computation, the two resulting metrics would also be relatively large, i.e., they would have positions near the head of the list. The search time associated with an ordered list is given by

$$t(\text{list search}) \approx (P)\,(t_c) \quad ,$$

where $P \leqslant T$ is the position of the desired entry. If the entry to be added is near the head of the list, then $P \ll T$, and the search time would be very small. A few simulations revealed, however,

* The data in column 1 of Fig. 17 reflect a slight modification of the original implementation and of the flow chart of Fig. 16: to speed up decoder operation when the table is full, more than one entry was discarded at Step G. The procedure was to find a metric value such that a fraction between f and f/2 of the entries on the table had metrics less than M, with f a specified small number. Although several searches of the table were usually required to find an M satisfying the criterion, on the average (3/4) (f) (T) entries were discarded each time Step G was performed, so that the time per discarded entry was low. This procedure, however, effectively reduces T by the factor $1 - (3/4)\,f$. For the data in column 1 of Fig. 17 [and Figs. 6(b) and 7(b)], $f = 0.1$ was used.

that for noisy messages, $P/T \sim (2/3)$, and therefore an ordered list offers little improvement. This result merely confirms the above-mentioned fact that during noisy messages the decoder spends most of its time investigating incorrect nodes which have $M_2 \leq M_1 < \text{BEST}$.

Another data structure that was considered was an ordered table. With this implementation, negligible time is spent performing Steps D and I of Fig. 16; most of the time is spent rearranging parts of the table each time an entry must be inserted (Steps C and H). As with a list, such an implementation is advantageous if the new entries are near the beginning (or near the end) of the table. Since this is not the case for noisy messages, the ordered table was abandoned.

3. Unordered Table with Higher-Order Searches

Finally, the structure shown schematically in Fig. 18 was chosen. The table of metrics is unordered but is divided into T_2 subtables of length T_1 each. The address of the largest metric on each subtable is recorded on a secondary table of length T_2. Thus, the i^{th} entry on the secondary table is the address of the largest metric on the i^{th} subtable. A corresponding secondary table is also needed for the addresses of the smallest subtable metrics. Finding the largest metric and replacing it with a new metric requires scanning the secondary table and

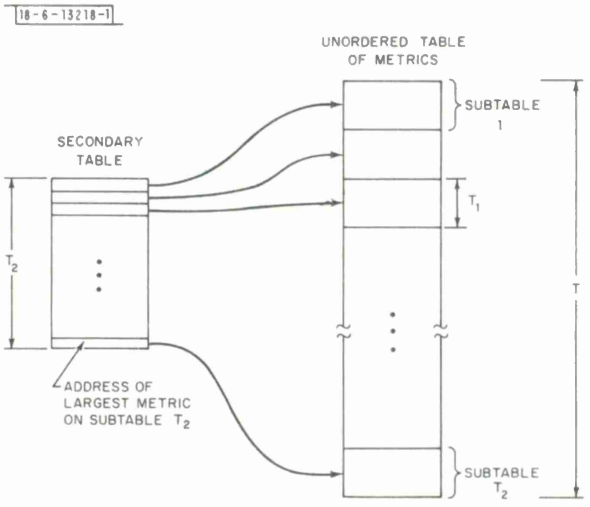


Fig. 18. Schematic representation of the second-order table data structure.

scanning one subtable. The search time is proportional to $(T_1 + T_2)$, and since $T_1 T_2 = T$, it is clear that the time is minimized if $T_1 = T_2 = \sqrt{T}$; the procedure is therefore termed a quadratic search with time given by

$$t(\text{quadratic search}) \approx (2T^{1/2})\,(t_c) \quad .$$

This concept may be carried farther with cubic, quartic, and higher-order searches possible. A cubic search, for example, would be used with a three-level structure: a metric table with T entries and $T^{2/3}$ subtables of length $T^{1/3}$; a secondary table with $T^{2/3}$ entries and $T^{1/3}$ subtables of length $T^{1/3}$; and a tertiary table with $T^{1/3}$ entries, one for each secondary subtable. The time required for a k^{th} order search is given by

$$t(k^{th} \text{ order search}) \approx (kT^{1/k})\,(t_c) \quad ,$$

and the extra memory needed for the auxiliary tables is

$$\text{extra memory} = 2\,[T^{(k-1)/k} + T^{(k-2)/k} + \ldots + T^{1/k}]$$

$$= 2\,\frac{T - T^{1/k}}{T^{1/k} - 1} \text{ words} \quad .$$

The factor of 2 included in this equation accounts for two sets of auxiliary tables for both the best and the worst metrics. Figure 19 shows the time and memory requirements for several values of k with T = 2048. Although the values do not include the additional operations that are necessary with larger k, the values are indicative of the enormous savings in time for k = 2, 3, or 4;

18-6-14432

ORDER OF STRUCTURE (k)	RELATIVE SEARCH TIME (t/t_c)	EXTRA MEMORY REQUIRED (words)
1	2048	0
2	91	92
3	38	348
4	27	714
5	23	1138

Fig. 19. Time and memory requirements for k^{th} order structures with T = 2048. The memory requirements include space for auxiliary tables with addresses of both the best and the worst metrics.

for example, k = 3 reduces the search time by a factor of more than 50. Furthermore, the extra memory required is small in comparison with the space occupied by the main table since, as discussed below, each entry in this table will have ~8 words. The second and third columns of Fig. 17 show the improvement in t_{DS} achieved by decoders using a quadratic and a cubic search, respectively; the data are derived from actual implementations on the Varian 620/i. The measured improvement is not as great as that predicted by the idealized data of Fig. 19 because (a) instructions other than those involved in t_c are associated with each data structure, (b) the percentage of these instructions increases as the total search time itself decreases, and – to a much lesser degree – (c) all subtable lengths are integral powers of two rather than the optimum value $T^{1/k}$. On the other hand, the value of t_{DS} in Fig. 17 is not quite the minimum because for $k \geqslant 3$ the subtable lengths become small enough to justify "opening" some of the search loops. That is, instead of looping through the same instruction sequence n times, the sequence is repeated n times and the instructions executed in succession. If this technique were incorporated in the cubic search decoder reported on in Fig. 17, a 10 to 15 percent improvement in t_{DS} would accrue. Furthermore, with T = 2048 a quartic search might yield a further reduction in t_{DS}.

4. Table-List Structure

The final data structure to be considered here is the one Jelinek used in his implementation[5] and is illustrated schematically in Fig. 20. This structure, which is a type of hash table, will be referred to as the table-list data structure. The node metrics are grouped into bins such that if a metric M falls in the range $k\Delta \leqslant M < (k + 1)\,\Delta$, then the entry with this metric is assigned to

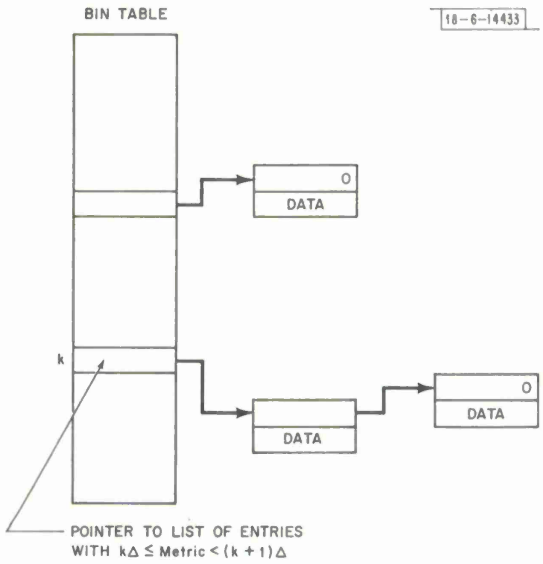


Fig. 20. Schematic representation of the table-list structure used by Jelinek. The data for the nodes are contained in the (unordered) lists pointed to by the BIN TABLE entries.

bin k. For speed of operation, Δ is chosen to be a power of 2. A table, denoted the BIN TABLE in Fig. 20, is used to store the addresses of the first entry in each bin, and all entries in the same bin are linked together to form a list structure,[11] i.e., each entry, in addition to containing the information pertinent to the decoding tree, also contains the address of the next entry in the bin. A zero address, for example, denotes the end of a list as well as an empty bin. The bin for a new entry is easily found by using the quantity (M/Δ) as an index. The best entry is taken from the highest non-empty bin, but this entry is not necessarily the one with the largest metric because the entries in a bin are not ordered. Similarly, the worst metric is taken from the lowest non-empty bin. Thus, the structure has many of the advantages of an ordered table, but also has a convenient means for inserting entries without rearranging large blocks of memory. There are, however, a number of drawbacks associated with this data structure. Additional memory is required for the BIN TABLE and for the address word contained in each list entry. Also, the parameter Δ must be chosen properly.* The ideal choice is $\Delta = 1$ in order to have exact ordering. But BIN TABLE must have $[M_{max} - M_{min}]/\Delta$ entries, where M_{max} and M_{min} are the maximum and minimum possible decoder metrics, so that $\Delta = 1$ (or any small Δ) results in excessive memory devoted to BIN TABLE. On the other hand, if Δ is large, the size of BIN TABLE decreases, but the metric ordering becomes inexact and decoder performance will deteriorate. Furthermore, the required size of BIN TABLE increases as the number of channel symbols increases because the possible range of metric values increases. However, by grouping all large negative metrics in one bin, the size of BIN TABLE can be reduced without affecting decoder performance.

In order to determine whether intermediate values of Δ degrade decoder performance, an elegant method of "simulating" the table-list structure with the already implemented data structure

*The table-list structure also lacks the flexibility that was needed in this study. For example, it is difficult to incorporate a procedure for discarding other than the entry with the smallest metric. Again, for purposes of this study, an implementation that employs exact ordering of metrics was necessary since such a decoder provides a performance standard for implementations using inexact ordering. Neither of these weaknesses is pertinent to a working version of the decoder.

was conceived. The method is to ignore (i.e., mask out) the $\log_2(\Delta)$ least significant bits when comparing metrics. Metrics that fall in the same width-Δ bin are thus treated as equal when the best or the worst metric is needed but, as in the actual table-list structure, the metrics retain their full precision when placed on the main memory table. This method therefore gives a very close approximation to the degradation in computation and in P_Q produced by the table-list structure. Simulations of this type were conducted with the effective Δ chosen so that the required size of BIN TABLE in an actual implementation would be ~500 words. The results show that at $R = R_{comp}$ and $T = 2048$, the table-list structure has performance comparable to a decoder with exact ordering in terms of (a) the average number of computations required, (b) the number of computations to achieve $P_Q = 10^{-3}$, and (c) the P_E. It was noted, however, that doubling Δ from the above value, i.e., halving the length of BIN TABLE, did increase P_Q appreciably. An estimate of the operations necessary with a table-list structure indicates that this implementation might reduce t_{DS} to ~150 cycles. The computation time based upon this estimate is shown in the final column of Fig. 17.

C. The Influence of Computer Hardware

As shown by the data of Fig. 17, the third-order table structure (and also the table-list structure) reduces the fraction t_{DS}/t_{COMP} sufficiently so that, in the implementations referenced by the figure, t_{PM} constitutes the largest portion of t_{COMP}, and effort must now be directed at reducing this quantity. Before this subject is pursued, however, it must be noted that the value of t_{COMP} is strongly dependent upon the instruction set and the hardware configuration of the computer being used. Although the times given in Fig. 17 are for the Varian 620/i, the relative magnitudes are probably typical for many small, general-purpose computers. On the other hand, the presence of certain features can make a computer much better suited to sequential decoding and can result in a significant reduction in decoding time.* As discussed in more detail below, an instruction that can determine the parity of a word will clearly reduce t_{PM}. A compare instruction might reduce t_{DS} by 5 to 10 percent, and a "decrement and jump" instruction (that decrements an index register and jumps if non-zero) would result in an additional reduction. Also, the presence of more than one accumulator would reduce t_{COMP} significantly. Because the 620/i has none of the above features, the times given in Fig. 17 can be reduced appreciably either by selecting (or designing) a computer better matched to the decoding problem, or by adding the desired features to an existing computer. Moreover, the hardware that is available will determine, to a large extent, the actual implementation that will be used.

D. Methods for Reducing the Time Spent Computing Parity and Metrics

In this section we shall show how t_{PM} can be significantly reduced, either with readily available hardware or with programming techniques. However, we first note that t_{PM} is roughly proportional to V, the number of parity networks. The first row of Fig. 21(a) shows the effect of halving V, from 12 to 6. The ~50 percent reduction in t_{PM} would reduce t_{COMP} by approximately 30 percent with the cubic table structure. It is important to realize, however, that

*The ultimate hardware option for the ZJ algorithm is an associative memory that enables the largest and the smallest metrics to be determined essentially by hardware rather than software. At the present time, however, such memories are not generally available and therefore they have not been considered.

18-6-14434

	t_{PM}(cycles)	
	V = 12	V = 6
ORIGINAL IMPLEMENTATION	920	480
WITH PARITY INSTRUCTION	720	380
WITH 2 ADDITIONAL ACCUMULATORS	620	320
WITH BOTH OF ABOVE FEATURES	480	260

(a)

	V = 12		V = 6	
	t_{PM} (cycles)	EXTRA MEMORY (words)	t_{PM} (cycles)	EXTRA MEMORY (words)
ORIGINAL IMPLEMENTATION	920	0	480	0
TABLE OF PAIRED METRICS	780	250	410	650
TABLE OF PRE-COMPUTED PARITY	460	2000	280	1000
BOTH OF ABOVE TABLES	310	2250	220	1650

(b)

Fig. 21. Estimates of the time required to compute parity and metrics if various improvements are incorporated. (a) Hardware improvements. (b) Programming improvements (including the additional memory that is needed).

this comparison was made with R/R_{comp} and not E_b/N_o held fixed. In fact, if E_b/N_o is fixed at the value necessary to achieve $R_{comp} = R$ with V = 6, then doubling V yields $R/R_{comp} = -0.14$ dB. Interpolating the data of Fig. 6(a), we find that this reduction in R/R_{comp} reduces by approximately 25 percent the number of computations necessary to realize a failure-to-decode probability of 10^{-3}. In this case, then, the total decoding time of a noisy block would be roughly the same for V = 12 as for V = 6, since the product (total number of computations) × (time/computation) is almost unchanged. However, other numerical values of the times t_{PM}, t_{DS}, and t_{MISC} could yield decoding times that for fixed E_b/N_o differed markedly for different values of V.

The most obvious instruction that would reduce t_{PM} is a parity instruction. On the 620/i, the parity of a word can be determined in 18 cycles, but a parity instruction might require only 2 cycles, and t_{PM} would be lowered by 20 percent. Similarly, if three accumulators were available instead of one, and if there were register-to-register arithmetic, t_{PM} would decrease by more than 30 percent. The presence of both of these features would result in an almost 50 percent time saving. (There is some overlap in the two calculated savings.) The above data are tabulated in Fig. 21(a). Note that the presence of extra accumulators will certainly reduce computing time in other parts of the program, although perhaps not to such a degree.

It is also possible to reduce t_{PM} by devising special programming techniques. A particular technique may eliminate the need for some hardware features or it may be used in conjunction with them to achieve even greater savings. Two especially profitable programming techniques will now be described; in each case, the decrease in t_{PM} is accomplished at the expense of an

increase in memory requirements. The values in the first row of Fig. 21(b) [reproduced from Fig. 21(a)] correspond to a set of instructions which does some initialization and then performs the following operations V times:

(1) Assume that a 0-bit was transmitted, and logically AND the (64-bit) shift register with the (64-bit) connection vector of one of the parity networks.

(2) Determine the parity of the result of (1).

(3) Look up the metric increment, using the result of (2) together with the quantized input signals to the decoder that correspond to this channel symbol. Also, look up the metric increment for a transmitted 1-bit. (The latter operation is simple since it depends only upon the result of (2) and upon the first bit of the connection vector.)

The first programming technique reduces t_{PM} by obtaining two metric increments simultaneously, so that Step (3) is performed only V/2 times per computation. The idea is to store the quantized signals by pairs rather than singly. Assume that there are 16 quantization intervals, so that $16^2 = 256$ interval pairs are possible, with each pair of quantized decoder inputs assigned a number between 1 and 256. The binary branch symbols (in the code tree) that correspond to a pair of received signals can take on one of the four values 00, 01, 10, or 11. A table containing $4 \times 256 = 1024$ entries is constructed, with one entry for each possible combination of transmitted and received signals; each entry in this table is the sum of two of the 16 metric increments. A 10-bit index built up from the two quantization interval numbers (2×4 bits) and the two branch symbols (2 bits) is then used to access the table. As an example, assume that the particular pair of received signals falls into quantization intervals $9 = 1001_2$ and $7 = 0111_2$. Then, if the corresponding branch symbols (for one of the two branches stemming from the current node) were 01, the index would be $(1001)\,(0111)\,(01)_2 = 605_{10}$, and the 605th table entry would contain the sum of the metric increments for a hypothesized 0-bit lying in interval 9 and a hypothesized 1-bit lying in interval 7. Thus, the original 16-entry table of metric increments is replaced by a much larger table, but the time spent performing Step (3) is now halved. Actually, the memory requirements do not increase by 1024 words because only half the storage is needed for the quantized input signals.* The net extra memory is thus $1024 - (1/2)(N + L)V$ words. The second row of Fig. 21(b) gives t_{PM} and the net memory requirements if this method is used with $N + L = 130$. The times can be decreased if some of the hardware features described above are also available. In fact, since Steps (1) and (2) are unchanged with this method, a parity instruction yields the same absolute saving and hence a larger percentage saving.

The second programming technique involves combining Steps (1) and (2) and using a large table to obtain all V parity bits at once.[12] Consider the proposed 64-bit decoder shift register to be made up of b-bit segments. Each of these segments can assume 2^b possible values, and for the set of V connection vectors, we can compute and tabulate, in advance, the V parity bits

*Actually, each quantized input occupies only $\log_2 Q \sim 4$ bits (i.e., the quantization interval number), and therefore several inputs could be stored in one 16-bit word. With the instruction set of the Varian 620/i, however, many operations are needed in order to recover a particular input if it shares a memory word with other inputs. Because these operations must be performed V times (or V/2 times with the above technique) for each decoder computation, and because the possible savings in memory were small [$< V(N + L)$ words] in comparison to the memory occupied by the other tables, one word was allotted to each input. On a computer with byte-oriented instructions, for example, the quantized inputs could be stored more efficiently.

that would result from each of these 2^b possibilities. During decoding, then, each of the (64/b) segments is used as an index into this table and the V parity bits corresponding to each segment are recalled with one indexed instruction. The V parity bits for a node are therefore obtained with (64/b) table look-ups together with a like number of exclusive-OR's – a considerably shorter process than performing Steps (1) and (2) V times. Now, the table corresponding to each segment requires $V2^b$ bits, so the entire table occupies (64/b) (V) (2^b) bits. On a 16-bit machine a convenient choice is b = 8, and a total of (2048) (V) bits is required. The third row of Fig. 21(b) shows that this procedure reduces t_{PM} by approximately 50 percent. For V = 12, one word is required for each entry, but for V = 6 two entries can share one word. At the expense of increased memory requirements, then, this technique not only obviates the need for a parity instruction but, because of the magnitude of V, actually yields a smaller t_{PM} than is possible with a parity instruction.

Finally, as indicated in the final row of the figure, the two techniques operating together yield further time reductions, but the memory requirements increase.

E. Some Design Considerations for a Real-Time Decoder

The data presented in the previous sections will now be used in a sample computation of the data rate and buffer storage requirements for a decoder operating in real time. Numerical results for some other examples are tabulated in Section VI.

Let us assume that some combination of the methods described above is used to reduce t_{PM} by 50 percent, and that a late-model computer with a cycle time of, say, 0.75 μsec is used.* Then a decoder using the table-list structure with V = 12 would require approximately 0.53×10^{-3} seconds for a computation during a noisy block. Now the decoder must perform an average of 2.4 computations per information bit (this figure is based upon a set of simulations with T = 2048 and C_{max} = 150 comp/bit), and thus the average data rate possible with the decoder is

$$[(0.53 \times 10^{-3} \text{ sec/comp}) (2.4 \text{ comp/bit})]^{-1} \approx 800 \text{ bits/sec} \quad .$$

Actually, this value is somewhat conservative since a computation performed during an average block requires less time than one performed during a noisy block because the decoder spends less time backtracking, and hence t_{DS} is smaller. At the above data rate, the decoder on the average keeps pace with the incoming signals, but when a noisy block is encountered, the decoder falls behind, and a buffer must be provided for storing the input signals until the decoder is ready to process them. (It is not necessary for this buffer to occupy high-speed memory.) In the worst case, a block will require 150 comp/bit, so that

$$(150 \text{ comp/bit}) (100 \text{ bits}) (0.53 \times 10^{-3} \text{ sec/comp}) \approx 8 \text{ sec}$$

of decoder time will be needed. If 4 bits are used for each quantized signal (Q = 16), four signals can be packed in one 16-bit word,† and if a tail of 28 bits is used, then at 800 bits/sec the buffer must contain a minimum of

$$(800 \text{ bits/sec}) \times (8 \text{ secs}) \times (12 \times 128/100 \text{ branch symbols/bit})$$
$$\times (0.25 \text{ words/branch symbol}) \approx 2.5 \times 10^4 \text{ words.}$$

---

* This is the cycle time currently available on the Varian 620/f, a computer very similar to the 620/i in all other respects.

† If more expedient for the decoder, the signals that comprise a message can be unpacked and stored one to a word when the message is to be decoded.

The above calculations are, of course, idealized because they do not allow for the statistical nature of the channel – in particular, the occurrence of a succession of noisier-than-average blocks. In order to allow for this possibility, the data rate must be less than the maximum possible and, if necessary, the buffer size must be increased. At a data rate significantly less than the maximum, the above buffer size will be adequate, but if the data rate is of prime importance, the buffer must be enlarged in order to reduce the probability of overflow (i.e., the probability that the allotted storage is inadequate). In general, the buffer size and the memory table for the ZJ algorithm would be adjusted so that the over-all failure-to-decode probability, viz., the buffer overflow probability plus the quit probability associated with finite T, is equal to 10^{-3}. The buffer size can be reduced by decreasing the values of V and Q, but such a reduction comes at the expense of an increase in the required signal strength.

It is worthwhile to note, however, that a ZJ decoder operating in real-time need never be idle.[5] If the most recent received signals correspond to depth d in the decoding tree, and if the current node is at depth d, then, while waiting for more received signals, the decoder can extend the table entry which has the largest metric and which is at depth less than d. This advance work costs nothing and, since it may be required at a later time, it permits operation at data rates closer to the maximum possible.

F. The Make-Up of a ZJ Table Entry

One final subject pertinent to the implementation is the make-up of an entry on the main table in memory. Since an entry corresponds to a temporarily discarded node, it must contain the information required by the decoder to resume decoding from that node. The essential information is the contents of the decoder shift register, the depth of the node, and the metric. In addition, a means must be provided for recovering the decoded information bits when the end of the tree is reached. For a decoder operating on long message blocks, the shift register length S is much less than the block length, and considerable storage can be saved if a separate list is maintained of the information bits that have passed through the shift register. The information sequence is then reconstructed from this list when decoding is complete. (One such scheme, although certainly not the only one possible, is described in Ref. 5.) For the decoder under consideration, however, N is relatively short and is of the same order as S, and the most straightforward scheme is also the most efficient in terms of storage: namely, keep the entire node name in an entry. (The node name for a node at depth d in the code tree is defined as the d-bit information sequence that leads from the start of the tree to the node.) The shift-register contents can be obtained from knowledge of the depth and the node name.

Now with the table-list data structure the most practical approach is to have all entries the same length, i.e., to reserve the maximum number of bits (N) for the node name in each entry. Entries of different lengths are difficult to handle with a list structure because of the problems that can arise when the allotted memory space becomes fully occupied. If the worst entry – the one that is discarded to make room for a new entry – is not of sufficient length to accommodate the new entry, then more than one entry must be discarded. But discarded entries are not generally located in contiguous core, so one long entry cannot be created from two short ones. The computation time needed to overcome this problem is unacceptably long.

With the k^{th} order table structure, however, entries of different lengths can be utilized to achieve worthwhile savings in storage. We shall consider one specific design on a 16-bit computer for a decoder with N = 100 bits. Eight words are sufficient for an entry: 1 word for the

metric, $\frac{1}{2}$ word for the depth (since $d < 256$) and $6\frac{1}{2}$ words for the node name. The first 8 bits of the node name share one word with d. For the main table, W words are allotted, and entries of either 4 or 8 words are permitted. A 4-word entry (referred to as a short entry) is used if $1 \leqslant d \leqslant 40$, and a long, or 8-word, entry is used if $d > 40$. Short entries are allocated from the top of the table, and long entries from the bottom. When decoding begins, the required entries will be short and will occupy positions at the top of the table but, as decoding proceeds, both short and long entries will be required, depending upon the depth of the entry. Eventually, if the message is noisy, all W words become occupied. This situation is illustrated in Fig. 22, in which j short entries and k long entries are indicated, with the demarcation between short and long entries denoted by A_d. Assume now that space must be found for a new entry, NEW. If NEW is of the same length as the worst entry, WORST, then NEW merely replaces WORST; if NEW is short and WORST is long, then WORST is discarded, $LONG_k$ is moved into the space previously occupied by WORST, NEW becomes $SHORT_{j+1}$, and A_d is incremented accordingly (and note that room is available for an additional short entry); finally, if NEW is long and WORST is short, then two entries are discarded (assuming the next-worst is also short), $SHORT_j$ and $SHORT_{j+1}$ are moved into the vacated space, NEW is inserted as $LONG_{k+1}$, and A_d is decremented accordingly.

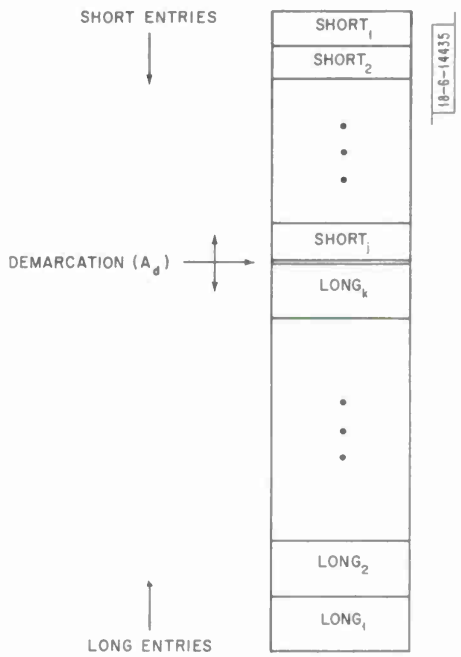


Fig. 22. Illustrating the use of different-length entries in order to achieve more efficient memory utilization (with k^{th} order table structure).

As exemplified by the metric of Fig. 3(b), a message that is difficult to decode generally has one bad noise sequence. This sequence, and hence the associated sharp dip in the correct metric, is equally likely to occur at any point in the message. If it occurs near the beginning of the message, most entries will be short when the decoding difficulty is encountered. Similarly, if it occurs near the end of the message, most entries will be long. The effective quit probability is determined by the number of entries that can be retained, and is thus roughly given by

$$P_Q|_{eff} \sim P\,[\text{metric dip at } d \leqslant 40]\; P_Q(W/4) + P\,[\text{metric dip at } d > 40]\; P_Q(W/8) \quad ,$$

in which $P_Q(T)$ is Pareto with exponent -1 at $R = R_{comp}$. Since there are few nodes near the start of the tree, entries with $d \leqslant 10$ will be ignored in the following computation, and since a path can be extended through the tail when $d = N$, the range of interest is $11 \leqslant d \leqslant 100$. With these substitutions, the above estimate becomes

$$P_Q|_{eff} \sim \frac{30}{90}\,\frac{1}{2}\,P_Q(W/8) + \frac{60}{90}\,P_Q(W/8) = \frac{5}{6}\,P_Q(W/8) \quad ,$$

which is 87 percent of the space required if all entries occupied 8 words. Actually, this factor can be reduced to 81 percent by using 5 words for the short entries (and there is no increase in

the complexity of the program).* Since $P_Q(2000) \approx 10^{-3}$, a total of $0.81(2000)(8) = 13 \times 10^3$ words is required for the node entries with a k^{th} order table structure. This compares with $(2000)(9) = 18 \times 10^3$ words for the entries with table-list structure, in which an extra address word is required in each entry.

VI. CONCLUSIONS

It has been shown that sequential decoding can be performed in real time at useful data rates on a small general-purpose digital computer. Techniques have been described for maintaining a very low error probability as the SNR decreases (at the expense of an increase in the quit probability). Suggested design features for a sequential decoder that operates efficiently (with respect to E_b/N_o) and that achieves high reliability are:

(a) Rate = R_{comp} at the nominal operating point.

(b) A long shift register ($\geq$64 bits) in order to reduce the error probability. The length of the tail is much less than that of the shift register.

(c) A computation cutoff, both to limit the decoding time for a message and (with the ZJ algorithm) to eliminate the possibility of an error occurring because of the finite memory.

(d) A threshold on the final metric value. A message is discarded if its metric falls below the threshold, thereby reducing the error probability at low SNR's. (Note, however, that with a long shift register, the first part of any decoded message, regardless of its metric, is very reliable.) The threshold is chosen to have a negligible effect on the quit probability at the nominal SNR.

For example, in order to meet the requirements $P_Q < 10^{-3}$ (at the nominal SNR) and $P_E < 10^{-7}$ (at any SNR) with the ZJ algorithm, the required table size is ~2000 entries, the computation cutoff is approximately 150 comp/bit, and the tail is approximately 28 bits. With binary antipodal signaling and a white Gaussian noise channel, the required E_b/N_o at $R = R_{comp}$ for a decoder with N = 100, L = 28, and Q = 16 is 2.7 dB if V = 12 and 2.8 dB if V = 6 (including the penalty associated with the tail).

The exact time and memory requirements of the decoder depend upon the method of implementation which in turn depends intimately upon the computer used. In order to present some quantitative estimates, let us assume that a Varian 620/f, having a memory cycle time of 0.75 μsec, is used; this computer is otherwise almost identical to the 620/i, upon which the data of Section V are based. Figure 23 gives pertinent information for three possible designs. The first design, which uses the table-list structure, is for maximum speed without regard to memory requirements,

---

* These estimates were partially confirmed by the following experiment with simulated messages that could not be decoded at $R = R_{comp}$ with T = 2048 entries. For each message, decoding was stopped at the instant when the correct path was about to be discarded. This is the critical moment, for if there were room for more entries at this stage, then decoding might be successful. The depth of each node in the memory table was recorded, and a probability distribution obtained, with the results

$$P(d \leq 40) \approx 0.19 \quad \text{and} \quad P(d \leq 56) \approx 0.41 \quad .$$

The latter value is of interest if a short entry is 5 words long. The values imply factors of 0.90 and 0.84, respectively, in comparison with 0.87 and 0.81 predicted by the heuristic arguments above.

DESIGN	FEATURE	REMARKS ON IMPLEMENTATION	V	DATA RATE POSSIBLE (bits/sec)	WORST-CASE DECODING TIME (sec)	MEMORY REQUIREMENTS (words × 10^3)
I	MAXIMUM SPEED	ZJ ALGORITHM; TABLE-LIST STRUCTURE	12	1000	6.3	21
			6	1100	5.5	20
II	INTERMEDIATE SPEED AND MEMORY	ZJ ALGORITHM; CUBIC TABLE STRUCTURE	12	600	10.5	16
			6	660	9.5	15
III	MINIMUM MEMORY	FANO ALGORITHM	12	300	21.	3
			6	370	17.	2

Fig. 23. Performance and memory requirements for three suggested designs. Values are for operation on a Varian 620/f, with $R = R_{comp}$, $P_Q = 10^{-3}$, and (for designs I and II) $T = 2048$.

the second design is an "intermediate" approach that uses a cubic table structure, and the final design minimizes the memory requirements by using the Fano algorithm. For the Fano decoder, a factor of 4 increase in the total number of computations (in comparison with the ZJ algorithm) has been assumed to meet the required $P_Q < 10^{-3}$. This value is an estimate gleaned from a number of published results on simulations with the Fano algorithm; Appendix D gives some justification for this estimate and an observation regarding an efficient implementation of the Fano algorithm on a general-purpose computer. With the Fano algorithm, then, the ~2000 entry table is not required, but the computation cutoff must be increased to approximately 600 comp/bit. All numerical values in the figure are for operation at $R = R_{comp}$. As noted in Section V-D, a small change in the ratio R/R_{comp}, produced, for example, by a small change in E_b/N_o or by a change in V with E_b/N_o fixed, can magnify into a substantial change in decoder performance.

In each of the three designs it is assumed that both the paired metric and the pre-computed parity programming techniques discussed in Section V-D are employed in order to reduce the time spent calculating the parity and metrics during each decoder computation. The data rates shown are the average that can be achieved for the particular implementation; they are valid for real-time operation only if all computer time is devoted to sequential decoding and only if sufficient buffer storage is provided for the incoming signals for messages that follow the one being decoded. In practice, the rate used would be somewhat less than the value shown. Also, it should be noted that if the time between receiving a message and decoding the message is critical, then the maximum data rate is dictated by the worst-case decoding time and is in fact equal to the reciprocal of that quantity. The worst-case decoding time is based upon the maximum allowed number of computations (i.e., the computation cutoff) and occurs with probability 10^{-3}. This time, together with the actual data rate and the values of V and Q, is used to calculate the required buffer size. The memory requirements that are given do not include this buffer because it need not be in high-speed memory. However, the requirements do include the storage needed for the paired metric and the pre-computed parity techniques and for the received signals (one per word) for the message being decoded. If either of the programming techniques is not used, the memory requirements will be reduced, but decoding speed will decrease, as detailed in

Fig. 21 of Section V-D. The space for the program itself (for any implementation) is less than 500 words and has therefore been neglected.

We see from the figure that approximately 20,000 words of memory are required with design I of Fig. 23, but a data rate on the order of one kilobit/sec can be achieved. Design II realizes ~60 percent of the data rate of design I but requires only ~70 percent of the memory. The data rate for this design can be increased perhaps by 15 percent by using a 4th-order (instead of cubic) table structure and by opening the code of some of the search loops, but the value for an actual implemented version is shown in order to provide a benchmark. With the Fano algorithm of design III, the memory requirements are minimal, and a data rate approximately one-third that of design I can be achieved. It should be noted that the data rates and decoding times for designs I and III are estimates although, as mentioned above, the values given for design II are accurate.

Each of the designs of Fig. 23 may be modified for operation at rates below or even slightly above R_{comp} by adjusting the time and (for the ZJ algorithm) the memory requirements accordingly. If, for example, $R/R_{comp} = -1$ dB (or, equivalently, if the nominal SNR is increased by approximately 1 dB), then the average data rate will approximately double, and the worst-case decoding time and the ZJ table size will decrease by an order of magnitude.

APPENDIX A
RANDOM NUMBER GENERATOR

Each noise sample used in the simulations was generated by mapping a uniformly distributed integer into a unique zero-mean normally distributed, or Gaussian, number.[13] The algorithm is fast and is very accurate, so that the quality of the Gaussian random numbers is limited only by the quality of the uniform numbers.

Considerable effort was expended in developing a satisfactory source of uniform random numbers. It was estimated that $\sim 10^6$ messages might be simulated, and since $\sim 10^3$ random numbers are needed per message, the period of the generator must be $>>10^9 \approx 2^{30}$. A simple linear congruential sequence[14] – the type most commonly used to generate uniform random numbers – can be conveniently generated on a 16-bit computer with a period of at most 2^{16}. Congruential generators are fast and easy to implement, and it was hoped that a combination of several of these generators would produce a satisfactory sequence with a long period. However, numerous types of combinations of congruential generators were tested, and none passed all criteria for randomness, so other generators were sought.

The algorithm that was finally developed generates a sequence $\{u_n\}$, $0 \leq u_n < 2^{15}$, of numbers according to the following relations:

$$x_n = x_{n-5} + x_{n-47} \qquad \text{mod } 2^{15}$$

$$y_n = 12589\, y_{n-1} + 7187 \qquad \text{mod } 2^{15}$$

$$u_n = x_n + y_n \qquad \text{mod } 2^{15}$$

Since $x^{47} + x^5 + 1$ is a primitive polynomial modulo-2, the least significant bit of $\{x_n\}$ is equivalent to the output of a maximum-length shift register sequence[15] and has period $2^{47} - 1$. The period of $\{u_n\}$ is therefore greater than $2^{47} - 1$. The congruential sequence $\{y_n\}$ is employed to overcome the possible occurrence of locally nonrandom subsequences in $\{x_n\}$. The sequence $\{u_n\}$ successfully passes the tests usually applied to uniform random number generators[14] (e.g., frequency test, serial correlation test), and in addition it passes a test that was found to be effective in weeding out unsatisfactory sequences: a chi-square test on the frequency distribution, applied to a long sequence of numbers ($>10^6$) and applied also to the χ^2-values of successive subsequences of the over-all sequence.

The additive sequence $\{x_n\}$ must be primed with a table of 47 integers. These integers were produced from the auxiliary pair of congruential generators

$$v_n = 22637\, v_{n-1} + 5801 \qquad \text{mod } 2^{15}$$

$$w_n = 15625\, w_{n-1} + 3 \qquad \text{mod } 2^{15}$$

combined according to the following algorithm.[16] The sequence $\{v_n\}$ is first used to fill a table with 32 entries. The 5 most significant bits of the next integer in $\{w_n\}$ are then used as an index into this table. The indexed entry is the output (i.e., it is stored in the table used to initialize the additive generator), and the entry is replaced with the next number from $\{v_n\}$. Although this appears to be an involved procedure for merely priming the table, the algorithm

is quickly executed on a computer and, more important, it was found that simpler initialization procedures led to unsatisfactory sequences $\{x_n\}$.

Three numbers – v_o, w_o and y_o – are thus needed in order to start the sequence $\{x_n\}$. If some set of x_i's starting from x_m are to be used again (e.g., those that correspond to a message that is to be simulated with different parameter values), then either the entire sequence of m numbers starting from x_1 must first be re-generated, or 48 numbers – x_{m-47}, $x_{m-46}, \cdots, x_{m-1}$, and y_{m-1} – must be available for initialization. To avoid this problem, a secondary sequence $\{u'_n\}$ of uniform numbers was used to periodically generate a triplet (v_o, w_o, y_o). The generator that produced $\{u'_n\}$ was of similar form to that used to generate $\{u_n\}$, but different multipliers and adders were used in the congruential generators, and the additive generator obeyed the equation $x'_n = x'_{n-41} + x'_{n-3} \bmod 2^{15}$. Thus, three numbers (v'_o, w'_o, y'_o) were used to specify the beginning of a set of simulated messages, and $\{u'_n\}$ was used to initialize the sequence $\{u_n\}$ at the start of each message. In order to reproduce the numbers for a particular message, the sequence $\{u'_n\}$ must be re-generated, but since there are $\sim 10^3$ random numbers per message, this operation is much faster than re-generating the sequence $\{x_n\}$.

APPENDIX B
TECHNIQUES FOR REDUCING THE MEMORY REQUIREMENTS

In order to reduce the table size required by the ZJ algorithm, a number of techniques were investigated. Three of the techniques yielded promising results, and they are described below. However, each of the techniques increases P_E – an effect that cannot be overcome without an appreciable increase in the required SNR when N ~ 100. However, the techniques may be of value in decoders with larger N, since then L can be increased without incurring much penalty.

The first technique is to decode messages in the backward direction as well as in the forward direction. This method is applicable because the encoding process is symmetric.* (Backward decoding is accomplished by reversing the connection vectors and reversing the decoded information sequence.) To test this procedure, 22×10^4 messages were simulated with R = R_{comp}, T = 2048, and C_{max} = 150. There were 312 quits in the forward direction, so that $P_Q = 1.4 \times 10^{-3}$. The decoder then attempted to decode these 312 messages in the backward direction and failed on 126, for an over-all P_Q of 0.57×10^{-3}. Thus, 60 percent of the forward quits can be successfully decoded in the backward direction. In fact, backward decoding is often accomplished with relatively few computations, so that this procedure can be used with the Fano algorithm (in preference to increasing the number of computations by 60 percent to reduce P_Q by 60 percent). Since there may be two decoding attempts on a message, P_E approximately doubles with this technique.

The second technique is an extension of the first, but is only applicable to the ZJ algorithm. The idea is to use some of the information obtained from unsuccessful forward decoding when decoding in the backward direction, as specified by the following procedure. Forward decoding is attempted, and if successful, produces a message as output. If, however, the decoder fails to reach the end of the decoding tree after C_{max} computation, forward decoding stops and the entries remaining in the table are examined. If all paths on the table agree in, say, the first A_{fwd} bits ($0 \leqslant A_{fwd} < N$), then backward decoding is initiated with an effective message length of $(N - A_{fwd})$ bits. Again, if backward decoding proves unsuccessful after C_{max} computations, but if the first A_{bak} bits of each path on the table are in agreement, forward decoding is reattempted. In this case, the first A_{fwd} bits are assumed correct and the effective message length is $(N - A_{bak} - A_{fwd})$ bits. Clearly, this procedure may result in an increase in P_E. In order to minimize this increase, a "safety factor" can be incorporated by disregarding B of the A bits when information is passed between decoder attempts. For example, the backward decoding attempt would use only MAX(0, $A_{fwd} - B$) bits from the first forward decoding attempt. Although no measurements of P_E were conducted, it is conjectured that the additional P_E introduced by this procedure will decrease exponentially with increasing B. Some empirical results obtained with this technique are shown in Fig. 24. The first row, for B = N = 100, is the quit probability when no information from forward decoding is used in backward decoding; these data correspond to the first technique described above. The remaining data show how P_Q decreases with decreasing B with the forward-backward-forward scheme. The significant reductions in P_Q imply an equivalent reduction in the required table size.

---

*Because all three techniques depend upon this symmetry, they are not applicable to the proposed long-shift-register decoder.

18-6-14437

B (bits)	OVER-ALL $P_Q(\times 10^{-3})$ AFTER DECODING	
	BACKWARD	FORWARD AGAIN
100	0.57	0.57
20	0.49	0.43
15	0.46	0.37
10	0.37	0.28
5	0.30	0.22
2	0.25	0.18
0	0.24	0.18

Fig. 24. Reduction in decoder quit probability obtained by using the forward-backward-forward decoding scheme, with information passed between successive decoder attempts. As described in the text, B represents a "safety factor."

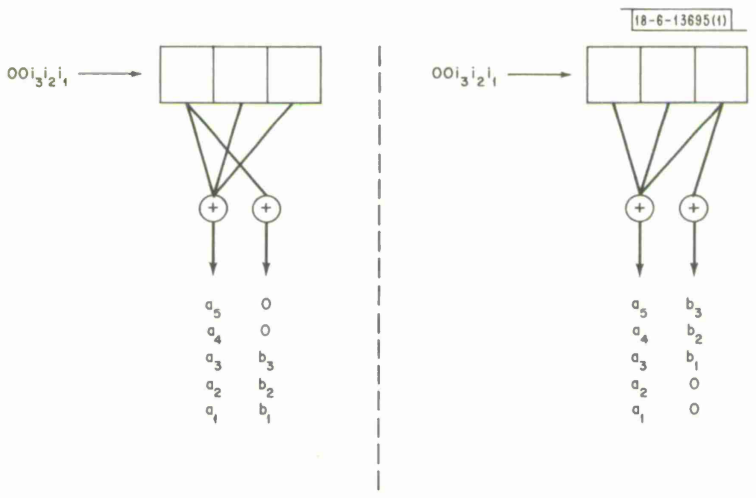


Fig. 25. Illustrating how a parity network can be shifted to obtain different combinations of received signals.

The final method requires the use of connection vectors c_i of different lengths so that if forward decoding fails, then the c_i and the corresponding received signals are appropriately shifted, and decoding is re-attempted.[17] As a simple example, consider the structure of Fig. 25, which has K = 3, but effectively has $K_1 = 3$ and $K_2 = 1$, where K_i denotes the constraint length of the i^{th} parity network. The sequence $\{i_j\}$ represents the information bits, the two 0-bits the tail, and $\{a_j\}$ and $\{b_j\}$ the received signals corresponding to the encoder output. The second part of the figure shows how the received signals become realigned when c_2 is shifted so as to be right-justified in the shift register. Thus, the advantage of this technique is that a different combination of received signals is created along each branch in the decoding tree, and decoding may be possible if some of the noisy combinations are broken up. The procedure yields even better results if the second decoder attempt is in the backward direction, because the reversed direction together with the shifted signals provides a trial that is highly independent of the original trial.

In order to determine the merits of this technique, a decoder was set up having V = 12 and having effective constraint lengths roughly uniformly spaced between the values 1 and 59. The average constraint length, defined by $\overline{K} = \sum^{V} K_i/V$, had the value $\overline{K} = 30$, so that the number of transmitted bits, $V(N + \overline{K} - 1)$, is identical to that of a decoder having all $K_i = 30$. The decoder attempted to decode 58×10^3 simulated messages at $R = R_{comp}$, T = 2048, $C_{max} = 150$, and failed on 67 messages, for a quit probability of 1.2×10^{-3}, which is comparable to that with equal K_i's. The $\{c_i\}$ were then shifted, and the decoder re-tried the 67 quits in the backward direction, with the result that 64 of the 67 forward quits were successfully decoded. In light of this striking improvement in P_Q, the simulated signal level was lowered so that operation was at $R/R_{comp} = +0.5$ dB. The forward quit probability was then 8.5×10^{-3} (based upon 68×10^3 messages), but backward decoding with shifted $\{c_i\}$ reduced P_Q by an order of magnitude.

Although this procedure appears very attractive, it has a serious drawback: the error probability is drastically increased. Initially, it was presumed that $P_E \sim 2^{-\overline{K}}$ at $R = R_{comp}$ Further investigation revealed, however, that the presence of unequal K_i introduces a different type of error mechanism,[18] namely, an increased probability that an incorrect path will remerge with the correct path after a specified number of nodes. Because of this, the error exponent for the $\{K_i\}$ used in the above experiment is not 30, but is 14 (Ref. 19). Also, the overall P_E increases by an additional factor of 2 because decoding may be performed in both directions. (If the $\{K_i\}$ were uniformly spaced between 10 and 60, maintaining $\overline{K} = 30$, then the exponent increases to 18 (Ref. 19), but the reduction in P_Q would then not be as great since the $\{c_i\}$ cannot be shifted as much.) This technique can also be combined with the preceding technique of passing information between forward and backward decoding attempts. In fact, many alternatives are possible: forwarded decoding with shifted $\{c_i\}$, backward decoding without shifting, etc. However, the method is not applicable unless $\overline{K}$ is substantially increased.

APPENDIX C
OTHER METHODS FOR REDUCING THE ERROR PROBABILITY

As mentioned in Section IV-C, two other methods for reducing the peak P_E were investigated and found to be less satisfactory than the long-shift-register technique. These methods are described below.

The first method was based upon the idea of detecting errors where their effect on the decoder metric is most pronounced. The curve sketched in Fig. 26 represents a threshold that

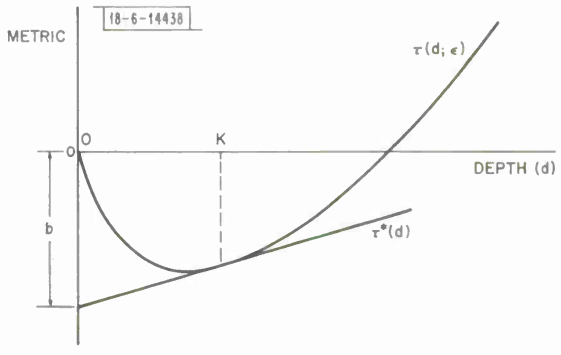


Fig. 26. Sketch of theoretical threshold curve $\tau(d;\epsilon)$. The probability that the path metric for the correct path falls below $\tau(d;\epsilon)$ is less than ϵ at any depth d.

could be applied to a metric at any depth in the decoding tree. If the node at depth d = 0 is correct, then the probability that the correct metric $M_c(d)$ falls below the threshold at depth d is less than ϵ. Thus, the threshold $\tau(d;\epsilon)$ is defined by

$$P[M_c(d) < \tau(d;\epsilon)] < \epsilon$$

and a path is immediately discarded if it ever falls below $\tau(d;\epsilon)$. Theoretical values for $\tau(d;\epsilon)$ which provide a good bound can be obtained.[20] Now a path that is correct at d = 0 but in error only at d = 1 will have a metric that on the average decreases for $1 \leqslant d \leqslant K$ and increases thereafter. If there is more than one error bit, the metric will tend to decrease for a longer span, but most incorrect paths fall below $\tau(d;\epsilon)$ near d = K. Thus, rather than use the actual curve if we use the tangent to τ at d = K, many error paths should be eliminated, but the likelihood of eliminating the correct path is considerably reduced. (Also, the implementation is simplified.) Furthermore, this linear threshold, denoted $\tau^*(d)$, is raised, as necessary, as progress is made in the decoding tree. Consider, for example, a path having the metric M(d) pictured in Fig. 27. The threshold is initially set at $\tau_0^*(d)$ which corresponds to M(0) = 0. Since M(1) < M(0), the threshold remains unchanged at this node, but M(2) is enough greater than M(0) that if this is the correct path — i.e., if the node corresponding to M(2) is correct — then the threshold should be raised to $\tau_2^*(d)$. The threshold τ_2^* remains in effect for this path until d = 5, at which point the metric has increased sufficiently to warrant the higher

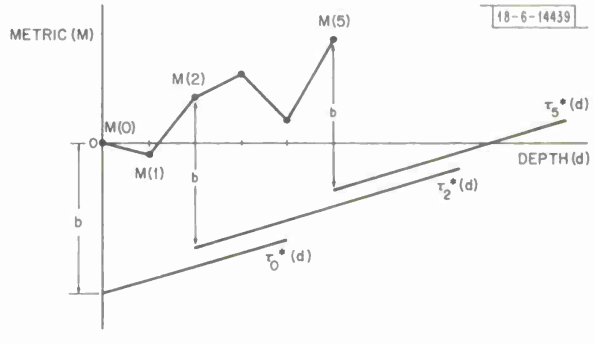


Fig. 27. Example of the use of a running (linear) threshold. A path is rejected if its metric ever falls below the most recent τ^* associated with the path.

threshold $\tau_5^*(d)$. Thus, associated with each path investigated is a threshold curve whose level is monotonically increasing, and a path is discarded if it ever falls below its current threshold. One parameter, the intercept (b in Figs. 26 and 27), is sufficient to specify the running threshold curve.

The above scheme was tried with the threshold chosen to roughly double P_Q at 0 dB. This strict threshold eliminated only about 1/3 of the errors at the worst-case SNR of −1.5 dB (cf. Fig. 13). Simulations were repeated with $\tau^*(d)$ obtained from different tangent points to $\tau(d; \epsilon)$, but the results were similar, indicating that this scheme is of limited value.

In the second technique, each message was decoded both forward and backward. A message was accepted only if decoding was successful in both directions and if the decoded messages agreed (and if the final threshold Θ was exceeded). This scheme reduced the peak P_E of Fig. 13 by approximately 2/3. In fact, all errors eliminated by the running threshold $\tau^*(d)$ were also eliminated by this forward-backward decoding. There is, however, an adverse effect on P_Q since a message must be decodable (viz., with $C \leqslant C_{max}$) in both directions; simulations indicate that this requirement increases P_Q by about 60 percent at 0 dB (cf. Appendix B), and therefore T must be increased by the same factor to maintain $P_Q < 10^{-3}$. Thus, the forward-backward scheme is preferable to the running threshold scheme since it has a greater effect on P_E and a smaller effect on P_Q, but there are still significant drawbacks, namely, the necessary increase in T and the increase in computation time because each message must be decoded twice.

APPENDIX D
REMARKS ON COMPARING THE FANO AND THE ZJ ALGORITHM

Since the literature contains experimental results from a number of studies with the Fano algorithm, it was hoped that a good estimate could easily be established of the comparative performance of the ZJ and Fano algorithms for the decoder parameters of interest here. Unfortunately, even though decoder performance depends primarily upon the ratio R/R_{comp}, most of the published results are for rate 1/2 decoders and for a binary symmetric channel, and hence there is enough of a difference between the parameters of the Fano decoders and the parameters of interest here that the accuracy of a detailed comparison would be questionable. Furthermore, from an examination only of rate 1/2 decoders with a binary symmetric channel, there emerges a wide range of values for the number of computations required. For example, the data given in Refs. 1, 21, and 22 show, respectively, that 300, 450, and 1000 comp/bit are necessary to achieve $P_Q = 10^{-3}$ at $R = R_{comp}$. (Part of this discrepancy may perhaps be attributed to the use of different code generators.) When compared with the ZJ requirement of 150 comp/bit,* these values represent computation ratios of 2, 3, and 6.7, respectively. Also, the comparative data given by Geist[23] of the relative computing time required by the ZJ and the Fano algorithms can be roughly converted to yield a decoder computation ratio of ~4, but the data of Jelinek[5] indicate a factor of 7 in the average number of computations.

Because of the above uncertainties, and because a Fano decoder offers an attractive reduction in the memory requirements, the Fano algorithm has been implemented with the same parameters used in the ZJ simulation. Preliminary results indicate that the computation ratio is approximately 4, both for an average block and for a worst-case (probability 10^{-3}) block, but further simulations are needed in order to accurately determine the performance of the algorithm.

In the course of implementing the Fano algorithm, it was noted that the decoding time could be reduced almost by half by the simple technique of storing in a table the two metrics that were computed at each node in the path from the start of the decoding tree to the current node. Since the net forward motion of the decoder is at most 100 levels into the decoding tree, if several hundred decoder computations are performed, roughly half the computations are performed when the decoder moves forward, and half when the decoder moves backward. With the above mentioned table, which occupies only 200 words, the relatively slow parity and metric calculation of each backward move is replaced by a fast table look-up operation, and the speed of the decoder is doubled. Since the preceding argument is valid for all but the quietest blocks, the effective computation ratio between the Fano and the ZJ algorithms can approximately be halved, making the Fano algorithm more competitive.

* This requirement is based upon a table size of ~2000 entries; as the table size increases, the computation requirement decreases, approaching ~100 comp/bit (cf. Section III-D).

ACKNOWLEDGMENT

Numerous informative discussions with I. Stiglitz are gratefully acknowledged.

REFERENCES

1. J. M. Wozencraft and I. M. Jacobs, Principles of Communication Engineering (Wiley, New York, 1965).
2. K. E. Perry and J. M. Wozencraft, "SECO: A Self-Regulating Error Correcting Coder-Decoder," IRE Trans. Inform. Theory IT-8, S128 (1962).
3. I. L. Lebow and P. G. McHugh, "A Sequential Decoding Technique and Its Realization in the Lincoln Experimental Terminal," IEEE Trans. Commun. Tech. COM-15, 477 (1967).
4. K. Sh. Zigangirov, "Some Sequential Decoding Procedures," Probl. Peredach. Inform. 2, No. 4, 13 (1966).
5. F. Jelinek, "A Fast Sequential Decoding Algorithm Using a Stack," IBM J. Res. Dev. 13, 675 (1969).
6. R. G. Gallagher, Information Theory and Reliable Communication (Wiley, New York, 1968).
7. K. L. Jordan, Jr., "The Performance of Sequential Decoding in Conjunction with Efficient Modulation," IEEE Trans. Commun. Tech. COM-14, 283 (1966).
8. I. M. Jacobs, "Sequential Decoding for Efficient Communication from Deep Space," IEEE Trans. Commun. Tech. COM-15, 492 (1967).
9. J. M. Wozencraft and B. Reiffen, Sequential Decoding (Wiley, New York, 1961).
10. M. Hellman, private communication.
11. D. E. Knuth, Fundamental Algorithms, Vol. 1 of The Art of Computer Programming (Addison-Wesley, Reading, Massachusetts, 1968), Chap. 2.2.3.
12. C. W. Niessen, "An Experimental Facility for Sequential Decoding," Technical Report 396, Lincoln Laboratory, M. I. T. (13 September 1965), DDC AD-631240.
13. I. Richer, "An Ideal Gaussian Random Number Generator for a Small Computer," to be published.
14. D. E. Knuth, Seminumerical Algorithms, Vol. 2 of The Art of Computer Programming (Addison-Wesley, Reading, Massachusetts, 1969).
15. W. W. Peterson, Error-Correcting Codes (M. I. T. Press, Cambridge, Massachusetts, 1961).
16. M. D. MacLaren and G. Marsaglia, "Uniform Random Number Generators," J. ACM 12, 83 (1965).
17. J. L. Massey, private communication.
18. E. A. Bucher, "Error Mechanisms for Convolutional Codes," Technical Report 471, Research Laboratory of Electronics, M. I. T. (August 1969).
19. E. A. Bucher, private communication.
20. I. Stiglitz, private communication.
21. C. R. Cahn and C. R. Moore, "DSCS Advanced Modulation System Study," Part II, Vol. IV, "Technical Support Data," MRL Report R-3003, Magnavox Research Laboratories, Torrance, California (June 1970).
22. J. L. Massey and D. J. Costello, Jr., "Nonsystematic Convolutional Codes for Sequential Decoding in Space Applications," IEEE Trans. Commun. Tech. COM-19, 806 (1971).
23. J. M. Geist, "An Empirical Comparison of Two Sequential Decoding Algorithms," IEEE Trans. Commun. Tech. COM-19, 415 (1971).

DISTRIBUTION

Chief of Naval Operations
Attn: Capt. R. Wunderlich (OP-941P)
Department of the Navy
Washington, D. C. 20350

Chief of Naval Research (Code 418)
Attn: Dr. T. P. Quinn
800 North Quincy Street
Arlington, Virginia 22217

Computer Sciences Corporation
Systems Division
Attn: D. Blumberg
6565 Arlington Boulevard
Falls Church, Virginia 22046
(3 copies)

IIT Research Institute
Attn: Dr. D. A. Miller, Div E
10 West 35th Street
Chicago, Illinois 60616

Institute for Defense Analyses
Attn: Mr. N. Christofilos
400 Army-Navy Drive
Arlington, Virginia 22202

Naval Civil Engineering Laboratory
Attn: Mr. J. R. Allgood
Port Hueneme, California 93043

Naval Electronics Laboratory Center
Attn: Mr. R. O. Eastman
San Diego, California 92152

Naval Electronic Systems Command
Attn: Capt. F. L. Brand, PME 117
Department of the Navy
Washington, D. C. 20360

Naval Electronic Systems Command
Attn: Mr. J. E. Don Carlos, PME 117T
Department of the Navy
Washington, D. C. 20360
(2 copies)

Naval Electronic Systems Command
Attn: Cdr W. K. Hartell, PME 117-21
Department of the Navy
Washington, D. C. 20360
(10 copies)

Naval Electronic Systems Command
Attn: Dr. B. Kruger, PME 117-21A
Department of the Navy
Washington, D. C. 20360

Naval Electronic Systems Command
Attn: Capt. J. V. Peters, PME 117-21
Department of the Navy
Washington, D. C. 20360
(2 copies)

Naval Electronic Systems Command
Attn: Mr. E. Weinberger, PME 117-23
Department of the Navy
Washington, D. C. 20360
(2 copies)

Naval Facilities Engineering Command
Attn: Mr. G. Hall (Code 054B)
Washington, D. C. 20390

New London Laboratory
Naval Underwater Systems Center
Attn: Mr. J. Merrill
New London, Connecticut 06320
(4 copies)

The Defense Documentation Center
Attn: DDC-TCA
Cameron Station, Building 5
Alexandria, Virginia 22314

DOCUMENT CONTROL DATA – R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Lincoln Laboratory, M.I.T.

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP
None

3. REPORT TITLE
Sequential Decoding with a Small Digital Computer

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Report

5. AUTHOR(S) *(Last name, first name, initial)*
Richer, Ira

6. REPORT DATE
24 January 1972

7a. TOTAL NO. OF PAGES
56

7b. NO. OF REFS
23

8a. CONTRACT OR GRANT NO. F19628-70-C-0230

b. PROJECT NO. 1508A

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
Technical Report 491

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
ESD-TR-72-32

10. AVAILABILITY/LIMITATION NOTICES
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Department of the Navy

13. ABSTRACT

Extensive simulations of a sequential decoder using the Zigangirov-Jelinek algorithm have been conducted on a small, general-purpose digital computer. These simulations prove that this type of computer has sufficient memory, sufficient speed, and sufficient flexibility to perform sequential decoding at useful data rates.

In this report, the memory and computational requirements of the algorithm are presented, and efficient methods for ensuring a very low probability of error at any signal-to-noise ratio (at the expense of an increase in the failure-to-decode probability) are discussed. The equations necessary to set up a decoder are given, and a number of possible computer implementations are suggested.

14. KEY WORDS

decoding
sequential decoding
Zigangirov-Jelinek algorithm
communications
simulation
small computer application